Konzeption und Realisierung einer Plattform für mobile und webbasierte Dienste auf Basis von J2EE unter Verwendung von freier Software

Diplomarbeit

im Studiengang Medieninformatik

zur Erlangung des akademischen Grades

Diplom Ingenieur - Medieninformatik (FH)

Fachhochschule Stuttgart, Hochschule der Medien, HdM

Betreuung

Prof. Walter Kriha

Dipl.-Ing. Tobias Frech, netads

Vorgelegt von Hans-Bernhard Friedrich am 23.02.2005

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfmittel verwendet habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hans-Bernhard Friedrich

Herrenberg, den 23. Februar 2005

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel und Motivation der Diplomarbeit

Die Firma *netads, Frech und Iffland GbR* betreibt einen mobilen und webbasierten Service auf PHP/MySQL-Basis, den sogenannten *SMSmonkey*. Für Kunden des SMSmonkeys ist es möglich, anhand einer von *netads* erstellten Java-Mobiltelefon-Applikation Textnachrichten per Mobiltelefon zu versenden und zu empfangen.
Ziel der Diplomarbeit ist es, den SMSmonkey-Service von PHP nach Java zu überführen und eine einfach zu erweiternde Plattform für weitere mobile und webbasierte Dienste zu schaffen. Der Schwerpunkt der Diplomarbeit liegt hierbei auf der Entwicklung der Plattform-Applikation, welche als Basis für zukünftige Services der Firma *netads* dienen soll. Als Referenz-Service der Plattform wird der SMSmonkey-Service auf der Plattform realisiert.
Motivation für die Umsetzung dieses Projekts ist der Wunsch von *netads,* alle ihre mobilen Services zentral, einfach zu warten und leicht erweiterbar unter einem Dach zusammengefasst zu haben. Dabei sollen die Schwierigkeiten, welche bei größeren Projekten mit der Verwendung von PHP als Programmiersprache auftreten, beseitigt werden, indem eine Java-Enterprise-Applikation entsteht.
Als ein Nebenziel ist die möglichst kostengünstige Entwicklung und der Betrieb der Applikation gefordert, was den aussschließlichen Einsatz von freier Software erforderlich macht. Diese zu erstellende Applikation wird „netads Mobile-Plattform“, kurz nM-Plattform genannt.

1.2 Der SMSmonkey

Ein Kunde des SMSmonkeys kann anhand einer von *netads* erstellten Java-Mobiltelefon-Applikation, einem sog. MIDlet, SMS zu günstigeren Konditionen versenden, als mit der schon im Mobiltelefon integrierten SMS-Funktionalität.
Dies ist möglich, weil das MIDlet den SMS-Text in einer HTTP-Verbindung per GPRS-Datenübertragung an den Server des SMSmonkey-Services sendet, welcher

wiederum den Service eines SMS-Gateway-Anbieters nutzt, der die SMS ins Mobilfunknetz leitet.

Die Kosten einer SMS, welche über einen SMS-Gateway-Anbieter versendet wird, sind deutlich günstiger, als der Versand über den ansonsten zu benutzenden Mobilfunk-Provider (z.B. T-Mobil, E-Plus). Diese Ersparnis gibt *netads* an seine Kunden weiter. In Summe mit den Kosten für die Datenübertragung per GPRS und der von *netads* erhobenen Gebühr für die Nutzung des SMSmonkey-Services kann, je nach Mobilfunk-Provider und Vertrag, beim Versand von SMS eine Ersparnis von 50% und mehr gegenüber dem herkömmlichen Versand, erreicht werden.

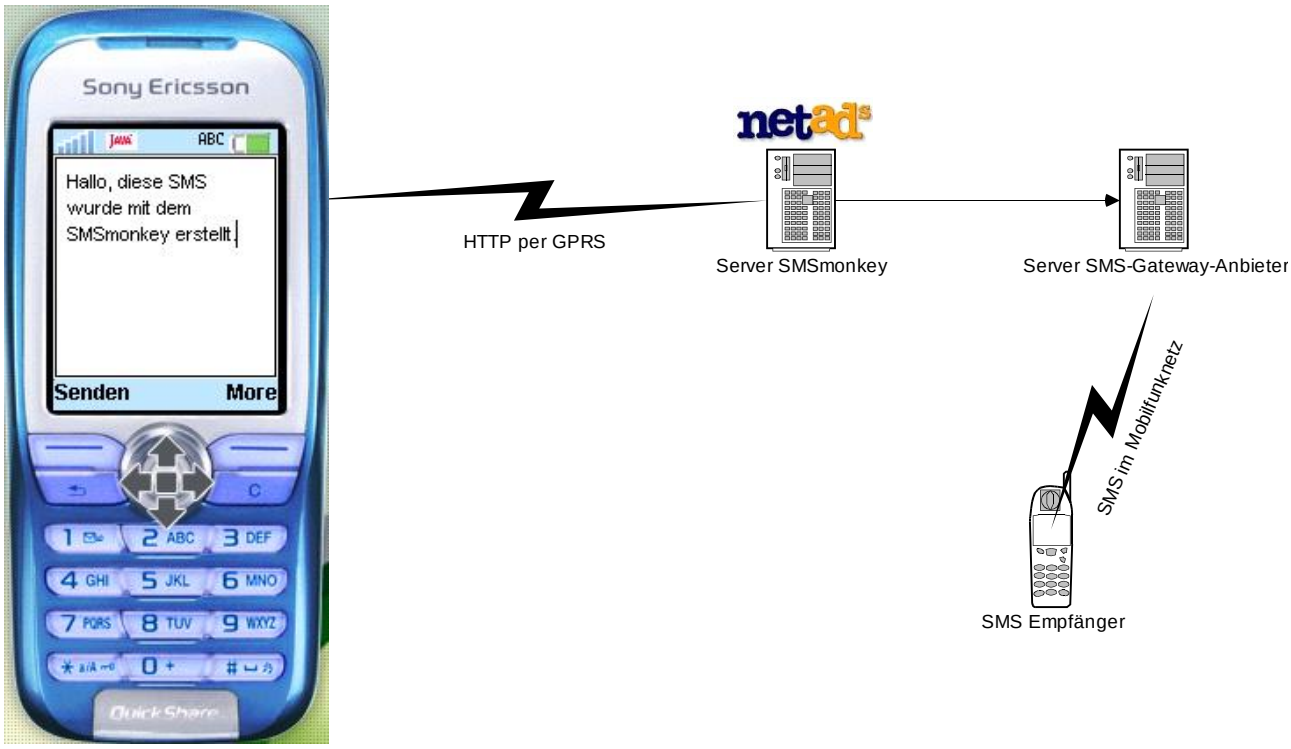

Abb. 1: Versand einer SMS mit dem SMSmonkey

Bei der PHP-Ausführung des SMSmonkeys stehen im Moment drei verschiedene Varianten von SMS zur Verfügung (Abb. 3):

- Standard

  Die für den Kunden günstigste Variante, wobei der Absender eine Nummer des SMS-Gateway-Anbieter ist.

- Deluxe

  Als SMS-Absenderkennung erscheint beim Empfänger die beim SMSmonkey-Service registrierte Mobilfunknummer des Senders.
- Anonyme SMS

  Als SMS-Absender erscheint der Text „Anonymous“. An das Ende des SMS-Textes wird noch *<anynomised by smsmonkey>* als Zusatz angehängt.

Abb. 2: Hauptmenü MIDlet

Abb. 3: Auswahl SMS-Typen

Alle SMS-Varianten können auch als Flash-SMS versendet werden. Wie der Name schon andeutet, wird anhand einer Flash-SMS eine Blitznachricht direkt auf das Display des Empfängers gefeuert.

Die anonyme Variante und die Tatsache, dass alle drei SMS-Varianten als Flash-SMS verschickt werden können, sind ein klarer Vorsprung an Features gegenüber dem SMS-Versand über die Mobiltelefon-eigene Applikation, welche Möglichkeit diese nicht bietet. Weiter besteht im PHP-SMSmonkey die Möglichkeit Textnachrichten, sog. Messages (abgekürzt MSGs), an andere Nutzer des SMSmonkey-Services zu senden (Abb. 2). Dies geschieht anhand des Store-and-Forward-Prinzips, ähnlich wie beim Versand von E-Mails. Die MSGs werden an den SMSmonkey-Service

geschickt und dort gelagert. Nimmt ein Kunde des SMSmonkeys über sein MIDlet Kontakt mit dem SMSmonkey auf, so bekommt er die an ihn gerichteten MSGs ausgeliefert. Die MSGs werden an das SMSmonkey-MIDlet übertragen und können auf dem Mobiltelefon gelesen werden. Da diese Art von Textnachrichten nicht in das Mobilfunknetz weitergeleitet werden und *netads* keine Gebühren für MSGs erhebt, ist der Preis für den Versand einer MSG für den SMSmonkey-Kunden sehr gering: Der Kunde hat nur die anfallenden GPRS-Datenübertragungskosten für die HTTP-Verbindung zum Server des SMSmonkeys zu tragen. Der Vorteil liegt auf der Hand: Sind Freunde und Bekannte eines SMSmonkey-Kunden ebenfalls Nutzer des SMSmonkeys, so kann untereinander per MSG kommuniziert werden. Im Vergleich zu herkömmlich versendeten SMS-Textnachrichten lassen sich so die Kosten für Textnachrichten-Kommunikation stark senken.

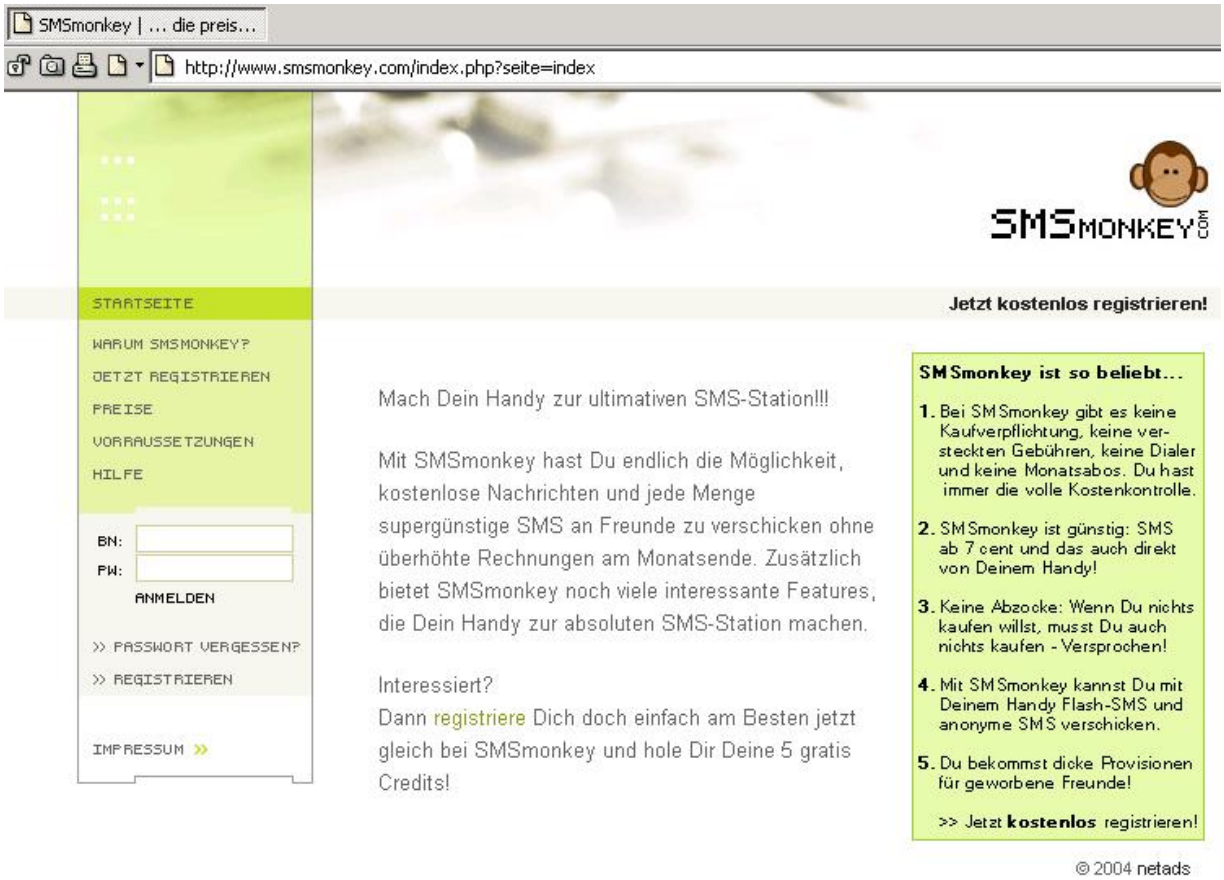

Abb. 4: Hauptwebseite SMSmonkey

Um den SMSmonkey nutzen zu können, muss sich ein potentieller Kunde über

dessen Webseite[1] registrieren (Abb. 4, Abb. 5), das SMSmonkey-MIDlet beziehen und für den Versand von SMS ein Paket mit Credits kaufen. Credits sind die Währung in der die Leistungen des SMSmonkeys bezahlt werden. Kauft der Kunde ein Paket mit einer bestimmten Anzahl von Credits, kann er diese für den Versand von SMS aufbrauchen. Nach einer einfachen Freischalte-Prozedur des MIDlets stehen alle Funktionen des SMSmonkeys zur Verfügung. In der PHP-Ausführung des SMSmonkeys ist der Versand von SMS und MSGs auch von dessen Webseite möglich, außerdem auch die Einsicht eines SMS-Postausgangs, sowie eines MSG-Posteingangs und -Ausgangs.

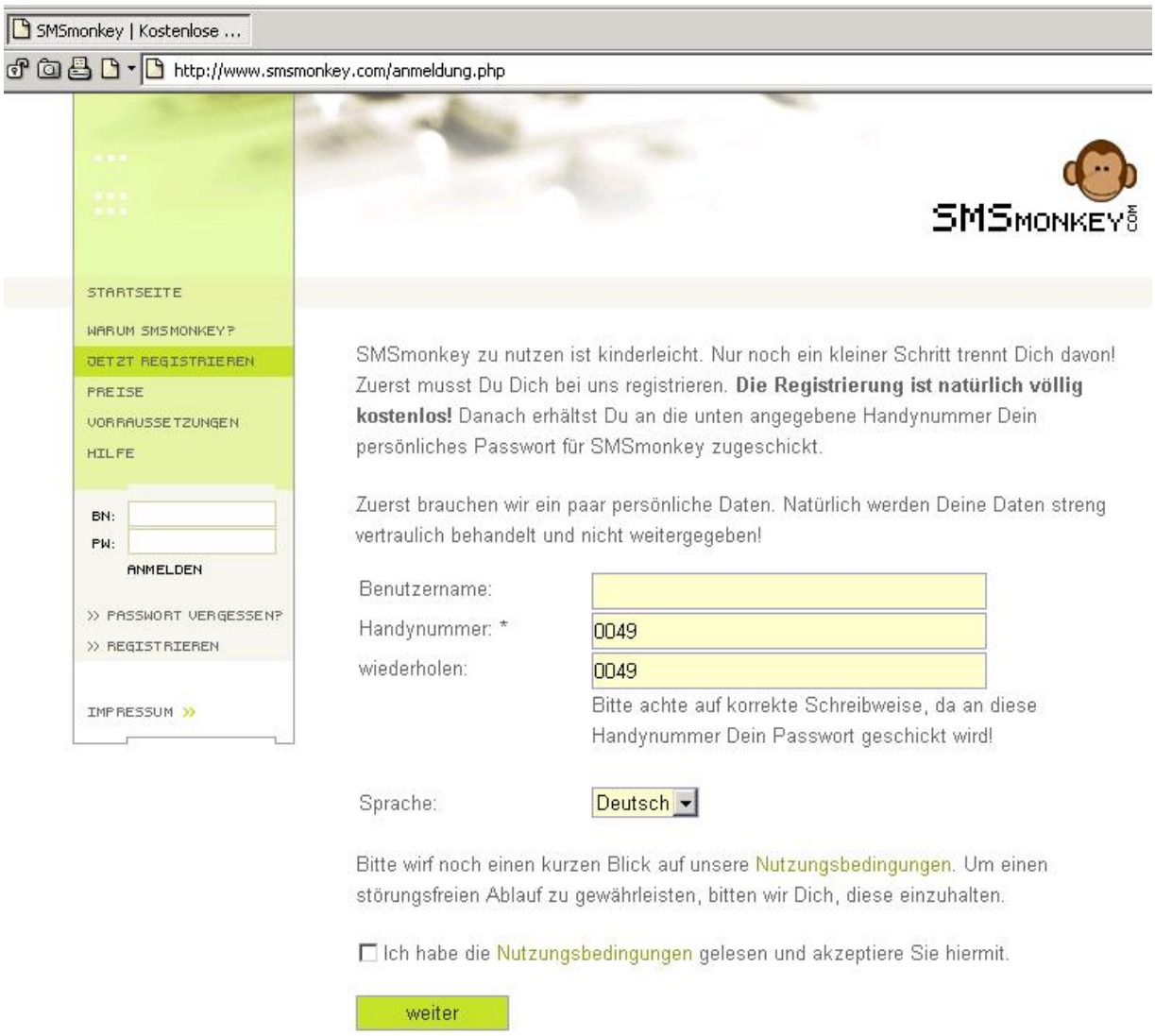

Abb. 5: Registrierung beim SMSmonkey

1 www.smsmonkey.com

2 Analyse

In diesem Kapitel sollen die Anforderungen an die zu entwickelnde Applikation der nM-Plattform und des SMSmonkey-Services dargestellt werden. Als erstes wird der SMSmonkey-Service betrachtet, dann die Plattform-Applikation. Von einem speziellen Service zu einer allgemeinen Plattform zu führen hat den Grund, dass ich genau diese Situation bei *netads* vorfand und diese Vorgehensweise meinem eigenen Lern- und Einarbeitungsweg entspricht. Dadurch erwarte ich ein besseres Verständnis des SMSmonkey-Services, der nM-Plattform und letztendlich des ganzen Projektes.

2.1 Analyse SMSmonkey-Service

Die Anforderungen an den SMSmonkey-Service der nM-Plattform gliedern sich in die Anforderungen an einen mobilen Teil und in einen webbasierten Teil. Der Schwerpunkt des SMSmonkeys liegt auf dem mobilen Anteil, da die Haupt-Serviceleistung das Versenden und Empfangen von Textnachrichten per Mobiltelefon ist. Der Web-Anteil dient vor allem der Anmeldung und Registrierung zum SMSmonkey-Service, der Pflege von Kundendaten und dem Erwerb von Credits.

2.1.1 Mobiler Service-Anteil

Abbildung 6 zeigt die grundlegenden Anforderungen an den SMSmonkey-Service aus Kundensicht unter Verwendung des mobilen Clients, des SMSmonkey-MIDlets: MIDlet freischalten, SMS versenden, MSG versenden/empfangen.

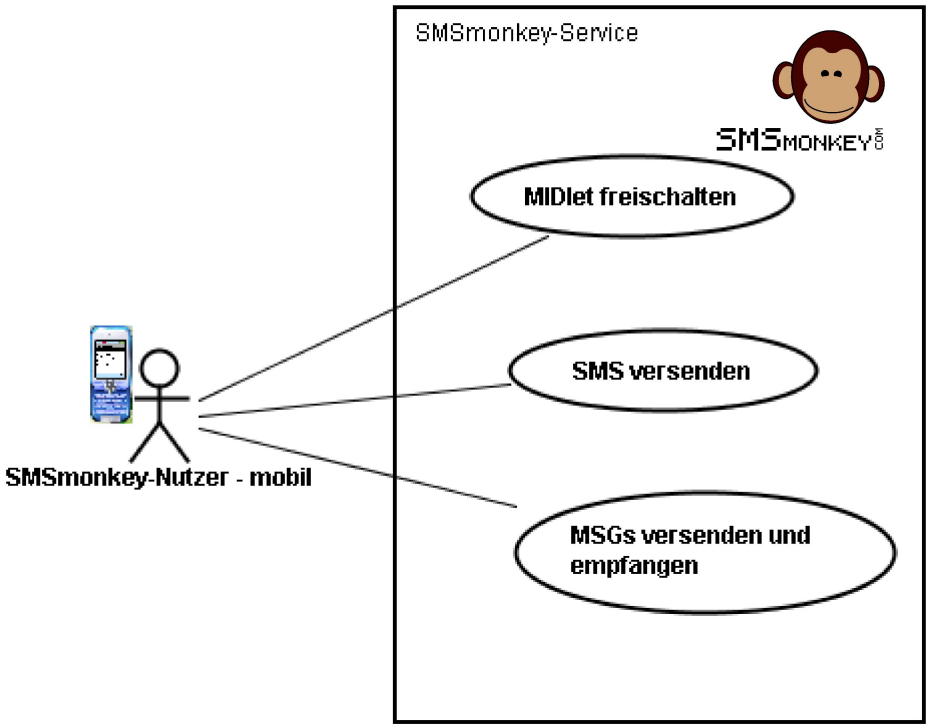

Abb. 6: Haupt Usecase SMSmonkey mobil

- freischalten

Dabei handelt es sich um den Anwendungsfall das SMSmonkey-MIDlet beim SMSmonkey-Service-System freizuschalten, um damit den mobilen Anteil des SMSmonkey-Service überhaupt nutzen zu können. Hierzu gibt der Kunde im SMSmonkey-MIDlet seinen Benutzernamen und sein Passwort an, und das MIDlet nimmt Kontakt zur nM-Plattform auf. Dort werden die angegebenen Daten verifiziert. Sind die Daten korrekt, kann der Kunde die Funktionen des MIDlets nutzen und SMS versenden bzw. MSGs versenden und empfangen.

- SMS versenden, MSGs versenden und empfangen

Nach der Freischaltung des mobilen Clients können Textnachrichten erstellt, versendet und empfangen werden.

Diese Haupt-Serviceleistung muss serverseitig im SMSmonkey-Service der nM-Plattform umgesetzt werden.

2.1.2 Webbasierter Service-Anteil

Um die in Abbildung 6 dargestellten mobilen Funktionen nutzen zu können, bedarf es einiger Vorleistungen des Kunden. Diese können auf der zu entwickelnden, webbasierten Infrastruktur erfüllt werden. Die Grundanforderungen an den Web-Applikations-Teil des SMSmonkeys gestalten sich wie in Abbildung 7 gezeigt.

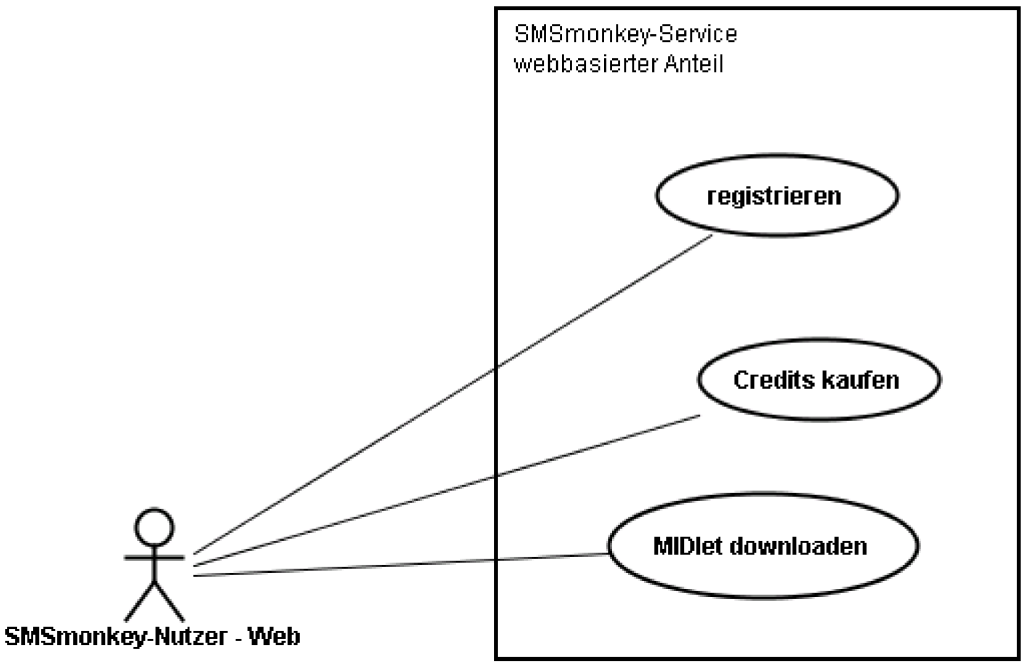

Abb. 7: Webbasierter Serviceanteil

- registrieren

Ein zukünftiger Kunde und Nutzer des SMSmonkey-Services muss sich auf einer Webseite unter Angaben von persönlichen Daten registrieren können, um die Leistungen des SMSmonkey-Services zu nutzen.

- Credits kaufen

Da bestimmte Leistungen des SMSmonkey-Services kostenpflichtig sind, muss der Kunde ein Paket mit Credits kaufen, um ein Konto-Guthaben herzustellen. Erst dann ist z.B. der Versand von SMS möglich. Die Web-Infrastruktur muss die Benutzerschnittstelle zu diesem Mechanismus zur Verfügung stellen, welcher in der Geschäftslogik des SMSmonkey-Services bzw. der nM-Plattform umzusetzen ist.

- MIDlet downloaden

Der Kunden muss in der Lage sein das SMSmonkey-MIDlet herunterladen zu können, um es auf seinem Mobiltelefon zu installieren.

Die genannten Anforderungen sind im SMSmonkey-Service umzusetzen und über eine Web-Infrastruktur zugänglich zu machen. Nutzt der Kunde die angebotenen Funktionen des Web-Anteils und erfüllt somit die geforderte Vorleistung, kann er den vollen Umfang des mobilen Anteils des SMSmonkeys nutzen.

2.2 Analyse netads Mobile-Plattform

Die netads Mobile-Plattform soll die Basis für den SMSmonkey und andere Services sein. Ein Service der umgesetzt werden soll, muss Schnittstellen der Plattform vorfinden, die er direkt benutzen kann um Unterstützung beim Erledigen seiner Aufgaben zu erhalten. Das Angebot an Schnittstellen sollte so groß wie möglich sein. Der Standpunkt, aus dem die Anforderungen gestellt werden, kommen aus der Sicht eines Programmierers, der die Plattform um einen neuen Service erweitern muss. Dies ist gleichbedeutend mit dem Standpunkt eines neuen, zukünftigen Services der nM-Plattform.

2.2.1 Plattform-Anforderungen

Anhand der Betrachtung des schon existierenden und in PHP gegossenen SMSmonkey-Services und der theoretischen Betrachtung von zukünftigen Services der nM-Plattform können Eigenschaften herausdestilliert werden, die für alle Services gelten und somit in die Anforderungen an die Plattform überführt werden können.

Allen umzusetzenden Services der nM-Plattform ist gemeinsam, dass sie Geschäfts- und Verwaltungslogik besitzen, die umgesetzt werden muss. Die Geschäftslogik ist in spezielle Leistungen des Services und Kunden- und Konto-Verwaltungslogik teilbar. Ein Service hat eine mobile oder eine webbasierte oder beide Arten von Schnittstellen

zu seinen Benutzern. Mit diesen Benutzerschnittstellen muss ein Service bzw. dessen Geschäfts- und Verwaltungslogik kommunizieren. Jeder denkbare Service der nM-Plattform besitzt eine Datenhaltung. Die zu haltenden Daten lassen sich in Daten speziell den Service betreffend und in allgemeine Verwaltungsdaten aufteilen. Zuletzt die umzusetzende Aufgabe oder eher Eigenschaft eines Services, zur Laufzeit konfigurierbar zu sein. Auch hier werden Werte verarbeitet, die Service-spezifischer und Service-übergreifender Natur sind.

Es ist zu erkennen, dass in den Geschäfts-, und Datenhaltungsbereichen eines Services stets allgemeine und Service-spezifische Aufgaben abgehandelt werden. Weiter teilt sich die Konfiguration von Applikations-Variablen ebenfalls in einen allgemeinen Teil und einen Service-Teil. Ziel ist nun die allgemeinen Belange der Services herauszulösen und an einer Stelle zusammenzufassen. Der Sammelpunkt dessen stellt der Bereich der nM-Plattform-Applikation dar, welcher im wörtlichen Sinne den Services als Plattform dient. Für die nM-Plattform bedeutet dies, dass sie die aus den Services herausgelösten Aufgaben zentral umsetzt und entsprechende Schnittstellen für die Services anbieten muss.

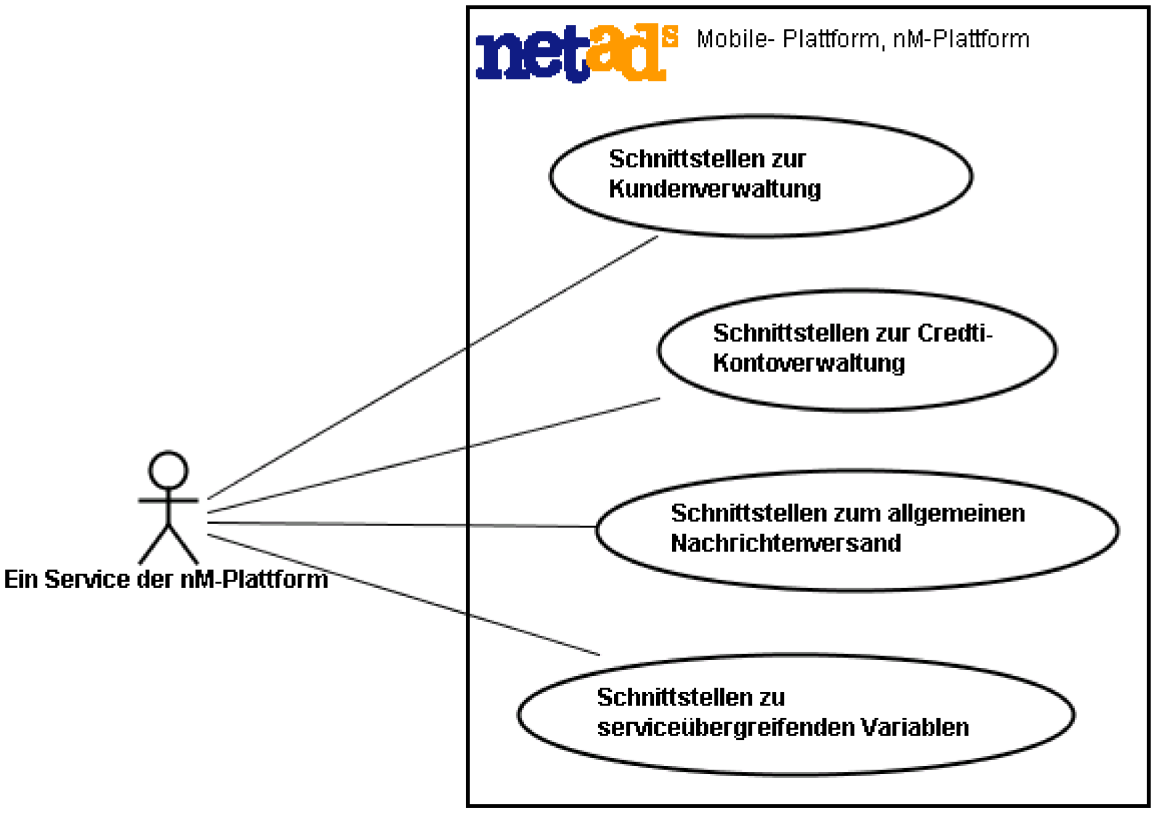

Abb. 8: Haupt Usecase der Plattform gegenüber einem Service

Es erwachsen Anforderungen an die nM-Plattform in der Form von in Abbildung 8 gezeigten Use-Cases. Hinter dem Angebot von Schnittstellen und Unterstützung für die Services steht die Umsetzung der dafür notwendigen Logik in der nM-Plattform-Applikation.

- Anbieten von Verwaltungsschnittstellen

Die nM-Plattform muss ihren aufsitzenden Services Schnittstellen zur zentralen, Service-übergreifenden Verwaltung von Kunden- und Kundenkontodaten anbieten können und den Services so viel Verwaltungslogik wie möglich abnehmen. Bei der Erweiterung der nM-Plattform um einen neuen Service soll so gewährleistet sein, dass jeweils nur eine kleine, Service-spezifische Verwaltungslogik umgesetzt werden muss, welche hauptsächlich als Adapter zur allgemeinen Verwaltungslogik dient.

- Anbieten von Schnittstellen zum allgemeinen Nachrichtenversand

Diese Arten von Schnittstellen sollen den Services ein für sie nutzbare Struktur anbieten, um ihren Kunden Informationen zukommen zu lassen. Gemeint ist hiermit

vor allem eine Schnittstelle zum Versand von E-Mails, deren Logik zentral umgesetzt ist. Der Versand von SMS zählt ebenfalls dazu, da dieser als Service-übergreifendes Kommunikationsmittel eingestuft ist, welches alle Services der nM-Plattform benötigen.

- Anbieten von Schnittstellen zu Service-übergreifenden Variablen und Konstanten

Um redundante Strukturen in den Services von vornherein zu vermeiden, muss die Plattform einen einheitlichen Zugriffsmechanismus für Service-übergreifende Werte anbieten und deren Haltung gewährleisten.

Eine weitere Art von Anforderungen an die Plattform sind Anforderungen in architektonischer Form. Anders ausgedrückt unterliegt die ganze nM-Plattform-Applikation gewissen architektonischen Richtlinien, welche in bestmöglicher Form umgesetzt werden müssen, um die Gesamtstruktur von Plattform und Plattformnutzern wahr werden zu lassen. Dazu zählt:

- Eine funktionale Modularität der Applikation
- Die Trennung von Zuständigkeiten
- Die Nutzung von Schichtarchitekturen
- Die lose Kopplung von Komponenten
- Reduktion von Komplexität: Nutzung einfacher Lösungen

Die Umsetzung dieser Richtlinien mündet in eine effiziente Architektur und kann an gewissen Qualitätsmerkmalen festgemacht werden. Für die nM-Plattform sind die wichtigsten Merkmale eine gute Erweiterbarkeit und Wartbarkeit der Applikation. Sie muss so gestaltet sein, dass ein neuer Service mit möglichst wenig Aufwand in die Tat umgesetzt werden kann.
Weitere qualitative Merkmale, die in der Umsetzung der nM-Plattform angestrebt sind, sind Zuverlässigkeit, Skalierbarkeit, Sicherheit und eine einfache Steuerung der gesamten nM-Plattform-Applikation.[2]

2 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

2.3 Aufgabenverteilung und Struktur

Die Anzahl der konkreten Anforderungen an den Plattform-Charakter der Applikation ist relativ gering. Die Anforderungen sind eher allgemeiner Natur, was aber die folgenden Arbeiten an Architektur/Design und Implementierung nicht erleichtert, sondern eher erschwert. Jede Anforderung für sich genommen eröffnet ein großes Feld an Architektur/Design-Überlegungen und mündet in einen umfangreichen Implementierungsaufwand. Die Umsetzung solch relativ inkonkreter, nicht richtig fassbarer Anforderungen birgt ein Risiko des Scheiterns eines Projektes in sich, da viele Weichen für das Projekt erst mit der Festlegung von Architektur und Design gestellt werden. Um dem vorzubeugen, die Analyse des Systems zu verfeinern und die Architekturentscheidungen zu erleichtern, wird eine nähere Betrachtung der Bereiche der nM-Plattform und deren Aufgaben durchgeführt. Diese soll die eigentliche Architektur vorbereiten und in diese münden.

2.3.1 Struktur

In Folge der Anforderungen läßt sich die nM-Plattform als Zusammenstellung verschiedener Sektionen, mit unterschiedlichen Aufgaben auf mehreren horizontalen Ebenen, wie in Abbildung 9, darstellen.

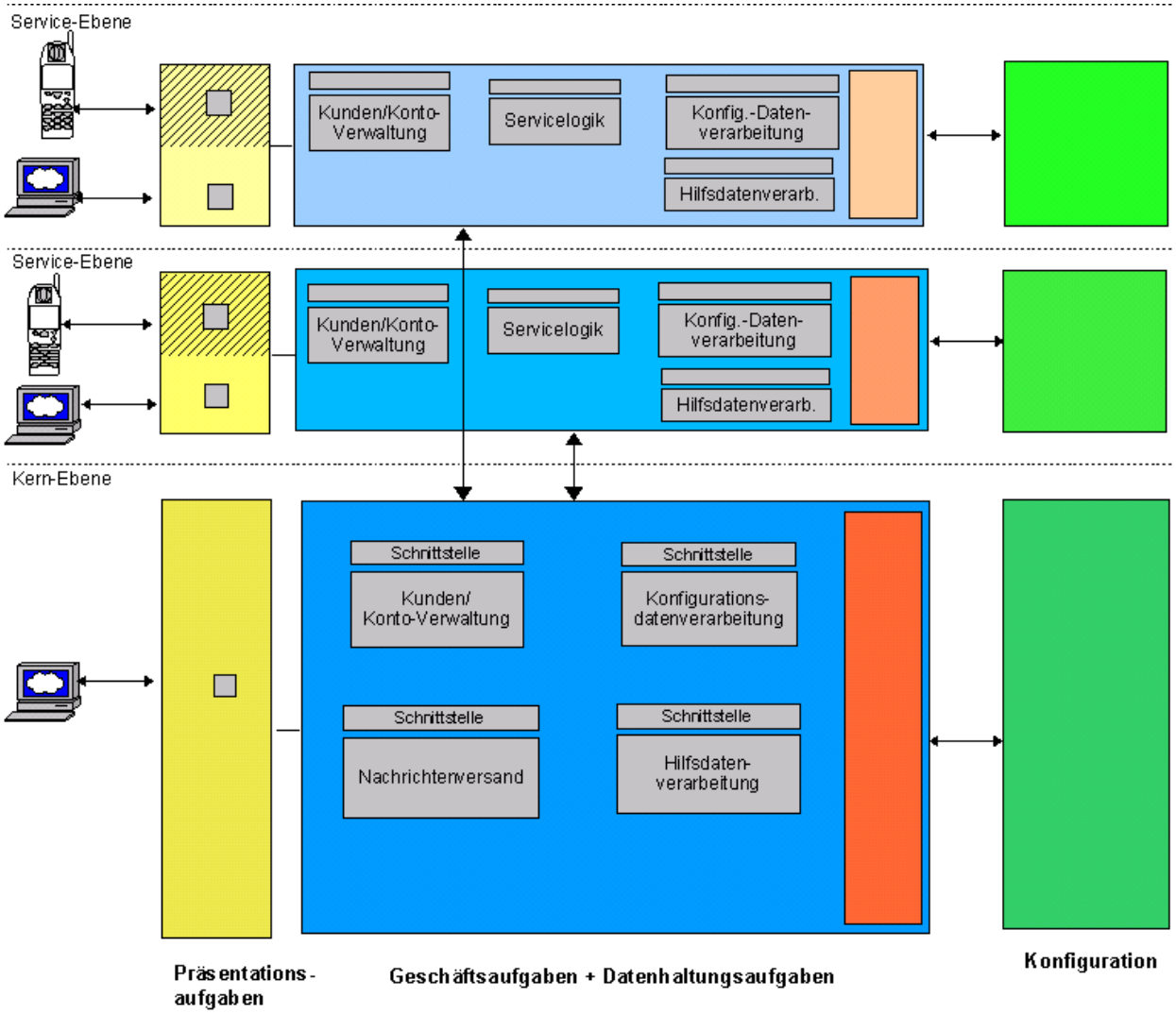

Abb. 9: Aufgabenbereiche und Ebenen der nM-Plattform

Die vertikalen Sektionen lassen sich in Bereiche für Präsentationsaufgaben, Geschäfts- und Datenhaltungsaufgaben, sowie der Konfiguration der nM-Plattform und ihrer Services aufteilen. Auf der untersten Ebene befindet sich die Kern-Ebene. Diese Ebene entspricht dem Applikationsteil mit Plattformcharakter, auf dem die Services aufsetzen und in dem alle allgemeinen Anforderungen und die ganze Applikation betreffende Aufgaben erfüllt werden. Die darüberliegenden Ebenen sind die Service-Ebenen, auf denen sich alles abspielt, was einen speziellen Service betrifft.

2.3.2 Aufgabenbereiche

- Geschäftsaufgaben

Im Geschäftsbereich der Kern-Ebene, der in Abbildung 9 dunkelblau dargestellte Mittelteil, werden die in Kapitel 2.2.1 genannten Schnittstellen-Anforderungen an die Plattform erfüllt. In diesem Teil existieren alle für die darüberliegenden Services nutzbaren Schnittstellen. Die Schnittstellen werden außerdem noch vom Präsentationsbereich, in Abbildung 9 gelb dargestellt, auf der selben Ebene benutzt.
Zu den Geschäftsaufgaben der Kern-Ebene gehören:

- Allgemeine Kunden- und Kontoverwaltung
- Allgemeiner Nachrichtenversand (z.B. Email-Versand)
- Bereitstellung von unterstützenden Daten für den Kern und die Services, z.B. von Hash-Werten über bestimmte Daten
- Kommunikation mit der die Kern-Ebene betreffenden Konfiguration und Verarbeitung der Konfigurationsdaten
- Benutzerdatenfilterung

Auf der Ebene der Services (hellblau) befindet sich in dieser Mittel-Sektion die Geschäftslogik des jeweiligen Services. Weiter werden an dieser Stelle kleine Verwaltungsaufgaben erfüllt, welche meist darin bestehen, die aus den Schnittstellen der Kern-Ebene bezogenen Kundendaten an die Bedürfnisse des jeweiligen Services anzupassen und umgekehrt. Wie die Kern-Ebene auch, benötigt jeder Service in gewissem Umfang eigene, unterstützende Hilfsdaten, die zum Erfüllen der Geschäftsaufgaben nötig sind, z.B. fortlaufende Nummern von Textnachrichten. Die Erzeugung dieser Hilfsdaten und anderer unterstützender Daten geschieht in dieser Sektion. Ebenfalls in diesem Bereich der Geschäftslogik befinden sich die Schnittstellen zur ausgelagerten Konfiguration des Services. Hier werden Datenumsetzungsaufgaben und Aufgaben zur Bereitstellung von Konfigurations-Daten erfüllt.

- Präsentationsaufgaben

Die Aufgaben im Präsentationsbereich auf Kern-Ebene (in Abbildung 9 dunkelgelb)

beinhalten das Bereitstellen einer allgemeinen Web-Infrastruktur. Dazu zählt zum Beispiel eine zentrale Login-Seite als Zugang zum Web-Angebot der einzelnen Services. Die Geschäftslogik hinter den Webseiten befindet sich im Geschäftsbereich des Kerns.
Auf den Ebenen der Services siedeln im Präsentationsbereich Schnittstellen in Richtung der Benutzer, welche Service-spezifische Präsentationsaufgaben erfüllen.
Diese Service-spezifischen Präsentationsaufgaben spalten sich in zwei Lager:

- webbasierte Leistungen eines Services und
- die Kommunikation mit den mobilen Clients des Services.

- Datenhaltungsaufgaben

Die in der Abbildung 9 in Orangetönen dargestellte Sektion der Datenhaltung erstreckt sich, wie alle anderen Aufgabenbereiche auch, über die Kern-Ebene und die Ebenen der einzelnen Services. Die einzelnen Datenhaltungsbereiche haben dem entsprechend unterschiedliche Verantwortlichkeiten und sind für die Datenhaltung der in der jeweiligen Ebene anfallenden Geschäftsdaten, zuständig. Im Kern sind dies Daten der Kunden- und Kontoverwaltung, Daten des Nachrichtenversands, sowie Konfigurations- und Hilfsdaten.
Auf den Ebenen der Services werden die speziellen Geschäftsdaten der Services persistent gehalten. Weiter werden spezielle Daten der Verwaltung eines Services und unterstützende Hilfs- und Konfigurationsdaten gespeichert.
In Abbildung 9 ist angedeutet, dass die Datenhaltungsaufgaben in den Geschäftsaufgaben aufgehen und zu den Geschäftsaufgaben zählen. Dies ist durch den später erläuterten Einsatz von CMP Entity Beans des J2EE-Frameworks begründet, welche dem Entwickler die Aufgabe der eigentlichen Datenhaltung abnehmen und verstecken. Logisch wird somit nur mit Objekten umgegangen, welchen die Eigenschaft persistent zu sein anhaftet. Allein diese Handhabung von Objekten stellt keine Umsetzungsarbeit einer Datenhaltung dar, warum die persistente Datenhaltung dem Geschäftsaufgaben-Bereich zugeschlagen ist.

- Konfiguration

Der Konfigurations-Teil der nM-Plattform und ihrer Services ist in Abbildung 9 in

Grün dargestellt. Der Konfigurations-Teil stellt eine separate Einheit dar, ja kann sogar als alleinstehende Applikation gesehen werden. Sie ist nur lose an die Geschäftsbereiche des Kerns und der Services gekoppelt.
Kommunikationspartner sind Schnittstellen im Kern-Geschäftsbereich, den Geschäftsbereichen der Services und dem entsprechende Klassen auf Seiten der JMX-Konfiguration, die diese Schnittstellen mit ausgewählten Werten bedienen oder aus ihnen Werte beziehen.

3 Techniken und Standards

Die vorangegangenen Analysen und die Einführung von Ebenen und Bereichen war noch losgelöst von den bei der Umsetzung der nM-Plattform eingesetzten Techniken. Mit der zunehmenden Konkretisierung in Architektur und Design in den folgenden Kapiteln wird das Wissen um Programmiertechniken nötig, weshalb im Folgenden auf die bei der Umsetzung der nM-Plattform eingesetzten Techniken und Standards eingegangen werden soll.

3.1 Die J2EE-Architektur

Das bei der Umsetzung der nM-Plattform maßgeblich eingesetzte J2EE-Framework spezifiziert im Wesentlichen eine Reihe von Interfaces, die vom Hersteller des einzusetzenden Application Servers implementiert werden müssen. Diese Vorgehensweise garantiert die Herstellerunabhängigkeit der Komponenten der Applikation. Der Entwickler kann sich auf eine Reihe festgelegter Interfaces, die eigentliche API, verlassen. Bei dem J2EE-Framework handelt es sich also nur um eine gemeinsame Schnittstelle für mehrschichtige, serverseitige Anwendungen. Der Entwickler kann sich voll und ganz auf die eigentliche Aufgabe konzentrieren (die Geschäftslogik) und muss sich nicht mit der Implementierung der Infrastruktur beschäftigen. Mit diesem Framework ist es möglich, serverseitige Komponenten zu entwickeln und zu kompilieren.[3]

3.2 J2EE-Komponenten

J2EE basiert auf einem Komponenten-Container-Modell. Vier Kern-Container liefern mit Hilfe ihrer jeweiligen APIs die spezifischen Laufzeitumgebungen, die für die verschiedenen Komponenten erforderlich sind.

3 vgl. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001

- Client-Komponenten

Eine J2EE-konforme Applikation kann entweder webbasiert oder nicht-webbasiert aufgebaut sein. Eine webbasierende Anwendung lädt Applets und HTML-Seiten auf einen Webbrowser herunter, was der Web-Infrastruktur der nM-Plattform entspricht. Eine nicht-webbasierende Applikation nutzt eine standalone Java Applikation, meist mit einer Swing oder AWT Benutzungsschnittstelle, als Client, ein sog. Application Client. Dieser greift direkt auf die Businesskomponenten (EJBs) zu. Es ist jedoch auch denkbar, aus einem Application Client heraus eine HTTP-Verbindung mit einem Servlet (Webkomponente) herzustellen. Der Container für Applets einer webbasierten Anwendung ist der Appletcontainer. In einem Application Client Container laufen die Application Clients ab.[4] Die mobilen Clients der Services der nM-Plattform, z.B. das SMSmonkey-MIDlet, können ebenfalls als Application Client betrachtet werden, sie laufen als eigenständige Applikation innerhalb der Java-VM des Mobiltelefons.

- Web-Komponenten

J2EE-konforme Web-Komponenten sind JavaServer Pages oder Servlets. Servlets sind Java-Klassen, die eine Anfrage dynamisch verarbeiten und eine Antwort erzeugen können. JavaServer Pages sind textbasierte Dokumente, die als Servlets ausgeführt werden. Diese beiden Arten von Komponenten laufen in einem Webcontainer eines Webservers ab.[4]

- Businesskomponenten

Die sog. Businesskomponenten dienen dazu, Geschäftslogik eines spezifischen Geschäftsbereiches umzusetzen. Die Businesskomponenten gemäß der J2EE-Spezifikation sind die Enterprise JavaBeans (EJB). Sie sind die Kern-Komponenten der J2EE-Spezifikation und laufen innerhalb eines EJB-Containers.[4]

3.2.1 Enterprise JavaBeans

Die nM-Plattform ist eine EJB-zentrische Applikation. Die am meisten eingesetzten Komponenten sind EJBs, und ihnen obliegt die Umsetzung der wichtigsten

4 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

Geschäftslogiken. In der nM-Plattform werden EJBs vom Typ Stateless Session Bean und CMP Entity Bean eingesetzt. Stateful Session Beans, Message Driven Beans und BMP Entity Beans werden aus Gründen der Vollständigkeit ebenfalls kurz erläutert.

Die Enterprise JavaBeans-Architektur ist eine von Sun Microsystems ausgearbeitete Standard-Architektur zur Entwicklung verteilter komponentenbasierter Unternehmens-Anwendungen auf Basis der Programmiersprache Java. Unter Verwendung von EJB-Servern, -Containern und -Komponenten können Applikationen erzeugt werden, die auf einem beliebigen EJB-konformen System zum Einsatz kommen können. Das EJB-Komponentenmodell ermöglicht eine Komplexitätsreduktion:
Die Verwaltung und Handhabung von Mechanismen wie Verteilung, Transaktionsverwaltung, Sicherheit etc. müssen nicht mehr vom Entwickler explizit programmiert werden. Die Komplexität dieser Aspekte wird in die EJB-Container und EJB-Server verlagert. Damit wird der Entwickler auf einem Gebiet entlastet, welches fehlerträchtig und aufwendig umzusetzen ist.
Eine Enterprise Bean wird überwiegend dazu genutzt, um spezifische Business-Logik zu kapseln. Die Instanzen einer Enterprise Bean werden zur Laufzeit in einem Container verwaltet. Die Laufzeitumgebung wird im Rahmen eines Applikations- bzw. EJB-Servers zur Verfügung gestellt. Ein solcher EJB-Server kann ein oder mehrere EJB-Container enthalten, die ihrerseits je eine EJB-Komponente mitsamt aller ihrer Instanzen verwalten. Die Steuerung und die Konfiguration von Enterprise Beans wird durch deklarative Anweisungen (Deployment Deskriptoren) ermöglicht, so dass kein Eingriff in die Komponenten selbst erforderlich ist.[5] [6]

3.2.2 Stateless Session Beans

Diese Sorte der Session Beans kann als eine Klasse mit nur statischen Methoden gesehen werden. Die Instanz dieser Klasse befindet sich auf dem Server und wartet auf Aufrufe. Die Stateless Session Beans sind nicht in der Lage, sich den Zustand zwischen den Methodenaufrufen zu merken. Sie sind also zustandslos. Der EJB-Container kann relativ leicht die Session Beans managen, da alle Instanzen einer

5 vgl. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001
6 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

Bean identisch und somit auch zwischen den Methodenaufrufen austauschbar sind. Stateless Session Beans können am schnellsten verarbeitet werden.[8]

3.2.3 Stateful Session Beans

Stateful Session Beans sind in der Lage, ihren Zustand zwischen den Methodenaufrufen, (meistens in ihren Instanzvariablen) pro Client zu merken. Es handelt sich dabei um ein so genanntes Konversationsgedächtnis. Nachdem ein Client eine Bean erzeugt hat, werden alle seine weiteren Geschäftsaufrufe zu der gleichen Instanz geroutet. Es wird also eine Art Session gestartet, die mit dem Beseitigen der Bean beendet wird. Die einzelnen Instanzen der gleichen Beans unterscheiden sich voneinander durch ihren Zustand. Die Beans sind somit nicht mehr zwischen den Methodenaufrufen austauschbar. Die Performance der Stateful Beans ist im Allgemeinen etwas schlechter als die der Stateless Beans. [7]

3.2.4 Entity Beans

Entity Beans verhalten sich grundsätzlich anders als Session Beans. Sie werden benutzt, um Geschäftsobjekte zu modellieren, die persistent sind. Entity Beans kann man an ihrem Namen erkennen: sie werden oft mit Substantiven benannt (Kunde, Bestellung, Rechnung etc.). Obwohl nicht unbedingt eine relationale Datenbank für die Persistenz der Beans sorgen muss, ist es trotzdem hilfreich, die Bean-Klasse als eine Tabelle und die Bean-Instanz als eine Zeile aus dieser Tabelle (die Daten) zu verstehen. Alle Operationen wirken sich persistent auf die darunter liegende Datenstruktur aus. Die Entity Beans lassen sich in folgende zwei Kategorien unterteilen:

- Bean Managed Persistence (BMP) Entity Beans

Falls der Entwickler sich für die BMP entschieden hat, muss er JDBC-Code schreiben, um die Persistenz der Bean sicherzustellen. Wie der Name schon andeutet, ist die Bean, also der Entwickler selbst, für die Umsetzung der Persistenz

7 vgl. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001

verantwortlich. Dies kann auf der einen Seite eine größere Flexibilität bedeuten, auf der anderen Seite ist die Nutzung von BMP Entity Beans fehlerträchtiger und aufwendiger.

- Container Managed Persistence (CMP) Entity Beans

In diesem Fall nimmt der Container dem Entwickler alle Verwaltungs- und Kommunikationsarbeit mit dem DBMS ab. Er muss sich nicht mehr darum kümmern und kann sich voll auf die Umsetzung des Business konzentrieren. Ein guter Container ist in der Lage, die Performance des Persistenzmechanismus zu optimieren und cached z.B. die Daten der Datenbank im Speicher, um die Anzahl der Datenbankzugriffe zu reduzieren.[8]

3.2.5 Message Driven Beans

Die Message Driven Beans (MDB) ermöglichen die asynchrone Verarbeitung von Nachrichten. Sie wird bei der Ankunft einer Java Message Service (JMS)-Message aufgerufen. Der Container wartet also auf die Ankunft von JMS-Nachrichten und ruft dann die entsprechenden Methoden auf.[9]

3.3 Die Neuerungen in der J2EE Version 1.3

Als die zur Umsetzung der nM-Plattform eingesetzte Version 1.3 der J2EE-Spezifikation von Sun im Sommer 2001 das Licht der Welt erblickte, fand bei den Enterprise JavaBeans ein großer Sprung statt. Der Löwenanteil der Neuerungen entfiel auf die Neueinführung der Message Driven Beans und auf die Fortentwicklung der Container Managed Persistence (CMP) für Entity Enterprise JavaBeans.
Neuerungen der Container Managed Persistence beinhalten eine bessere Unterstützung der automatischen persistenten Abbildung von Enterprise Beans auf relationale Datenbanken.

8 vgl. Mark Wutka, J2EE Delevoper's Guide, Markt&Technik, 2001
9 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

Die wichtigsten Erweiterungen sind:

- Objektbeziehungen (Relationships) zwischen persistenten Beans
- Eine SQL-ähnliche Abfragesprache (EJB-QL)
- Lokale Interfaces, die den mit Remote Interfaces verbundenen Aufwand vermeiden.

3.3.1 Objektbeziehungen (Relationships)

Dies ist ein interessantes Feature der Neuerungen, da es das CMP-Persistenzmodell nahe an das Niveau von echten objekt-relationalen Mapping-Tools bringt.
Beziehungen zwischen Beans müssen nicht mehr von Hand ausprogrammiert werden (Lookup des Home-Interfaces, Aufruf einer Finder-Methode, ...), sondern werden deklarativ im Deployment Descriptor definiert. Relationships werden zwischen Beans deklariert, die über Local Interfaces verfügen, sind zweiseitig navigierbar, und der Container kümmert sich voll und ganz um die Konsistenz der Beziehung der beteiligten Beans.

3.3.2 Local Interfaces

Dieses Feature ist nicht auf Entity Beans beschränkt, sondern kann auch auf Session Beans angewendet werden. Ein Local Interface wird ähnlich wie die bisherigen Remote Interfaces implementiert und im Deployment Descriptor deklariert. Local Interfaces definieren eine Schnittstelle zu dem Bean, die nur innerhalb derselben Java Virtual Machine aufrufbar ist. Dies ermöglicht es, bei der Parameterübergabe auf die Serialisierung und Deserialisierung der Parameterwerte zu verzichten – die Übergabe erfolgt mittels Pass-By-Reference, wie bei normalen Java-Methodenaufrufen. Ein Bean kann sowohl über ein Local als auch über ein Remote Interface verfügen.

3.3.3 EJB-QL

EJB 2.0 definiert eine eigene Abfragesprache für Container Managed Persistence. Diese Sprache ist eng an SQL angelehnt – den Standard für relationale Datenbanken. Sie erweitert bzw. modifiziert diesen aber mit Blick auf die besondere Semantik von persistenten Objekten. Insbesondere können Attribute mit einer Java-ähnlichen Syntax angesprochen werden. EJB-QL macht die Formulierung von Finder-Methoden unabhängig vom eingesetzten DBMS. Es stellt quasi ein Meta-SQL dar. Finder-Methoden müssen somit beim Wechsel des DBMS nicht bearbeitet werden, da sie keinen proprietäre SQL-Code enthalten.

Zur Thematik dieses Kapitels, den Neuerungen in der J2EE Version 1.3, vgl. Javamagazin 03.02, Software&Support, 2002.

3.3.4 Servlet Filter

Seit der Java Servlet API 2.3 der J2EE-Spezifikation 1.3 wurde ein neues, mächtiges Konzept in die Java Servlet API integriert: *Java Servlet Filter*, welche ebenfalls in der nM-Plattform zum Einsatz kommen und im Zuge der vorangegangenen Standards vorgestellt werden sollen.

Dieses neue Konzept der Java Servlet Filter ermöglicht es, Komponenten zu programmieren, die sich einfach und dynamisch in die Verarbeitungskette zwischen Client und Servlet integrieren lassen (Abb. 10). Es können Aufgaben, die nicht direkt mit der Logik des Servlets zusammenhängen, von selbstständigen Komponenten erledigt werden, die sich einfach hinzufügen und gegebenenfalls wieder entfernen lassen. Dafür muss kein Code des Servlets geändert werden. Es genügt die Filter, die dem Servlet zur Seite stehen sollen, in der entsprechenden Servlet-Container-Registrierungsdatei (*web.xml* bei Tomcat) zu registrieren.

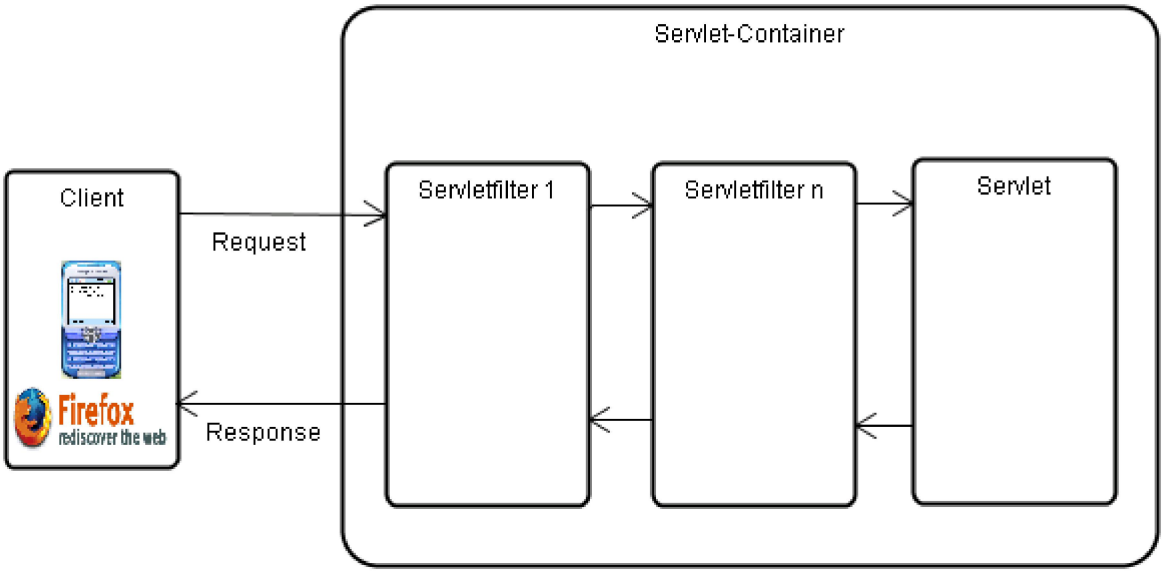

Abb. 10: Verarbeitungskette mit Servletfiltern

Ein Filter kann auf einfache Art die Kommunikation zwischen Servlet und Client beeinflussen. Der Filter hat Zugriff auf die Headerinformation und auf die Daten der Request- und Response-Pakete, die zwischen Client und Servlet hin und her geschickt werden. Dadurch sind sie geeignet, gewisse Aufgaben in verschiedenen Bereichen wie Authentifizierung, Verschlüsselung, Daten- Kompression und -Transformation zu erledigen. Das Konzept, das den Java Servlet Filtern zugrunde liegt, ist eine Implementation des Chain of Responsibility Patterns.[10] [11]

In der nM-Plattform kommt ein Servletfilter zum Einsatz, um die Parameter eines HTTP-Requests, welcher an das für die Kommunikation mit den mobilen Clients zuständige Servlet gerichtet ist, auf Konformität an erwarteten Datentypen und auf verbotene Zeichenketten untersucht. Wird im Servletfilter bei diesen Prüfungen ein Fehler entdeckt, sendet der Servletfilter direkt eine Meldung in Richtung des Clients, ohne dass der fehlerhafte Request im darauffolgenden Servlet oder tiefer in den Geschäftskomponenten Schaden anrichten kann.
Bei der Bereitstellung einer zentralen Authentifizierung über eine Login-Seite kommt ebenfalls ein Servletfilter zum Einsatz. Der Servletfilter überprüft die an die Web-Infrastruktur gerichteten Requests auf Vorhandensein eines Attributs, welches die

10 vgl. http://www.cs.fh-aargau.ch/~gruntz/courses/sem/ss03/filters.pdf, 20.01.05
11 vgl. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001

erfolgreiche Anmeldung eines Kunden am System dokumentiert. Fehlt dieses Attribut, wird der Request an eine Login-Seite umgeleitet.

3.4 Struts

Der Fokus bei der Entwicklung der nM-Plattform lag in der Umsetzung der Plattform-Applikation. Trotzdem war gefordert, den Web-Applikations-Teil in einer rudimentären Form anzulegen, um ein minimales Web-Frontend der nM-Plattform, bzw. des SMSmonkey-Services zu Testzwecken zu haben. Von diesem Stand aus sollten die webbasierten Belange der nM-Plattform schnell auszubauen sein. Der Web-Anteil der nM-Plattform und ihrer Services wird mit dem durch seine Beliebtheit und Flexibilität bekannten Apache-Web-Framework *Struts* realisiert.

Struts ist ein MVC Model 2 basiertes Open-Source-Framework für die Präsentationsschicht von Webanwendungen. Es besteht aus kooperierenden Klassen, Servlets und JSP-Tags. Das Framework läuft auf einem JSP- bzw. Servlet-tauglichen Webcontainer wie z.B. Tomcat. Durch die Einführung des MVC Model 2-Konzepts trennt das Framework die Komponenten Model, View und Controller transparent voneinander und stellt dem Entwickler komfortable und einfache Schnittstellen zur Realisierung von Web-Applikationen zur Verfügung. Neben dieser komponentenbasierten Architektur ermöglicht das Framework die Ressourcen effektiv einzusetzen: Zum Einen benötigen die JSP-Entwickler keine Java-Kenntnisse, um die JSP-Seiten zu implementieren, zum Anderen übernimmt der Standard-Controller die Konvertierungen des HTTP-Protokolls, so dass sich die Entwickler auf die Kernaufgaben konzentrieren können.[12] [13]
Abbildung 11 zeigt eine Übersicht der Struts-Komponenten und deren Zusammenspiel.

12 vgl. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001
13 vgl. Javamagazin 04.02, Software&Support, 2002

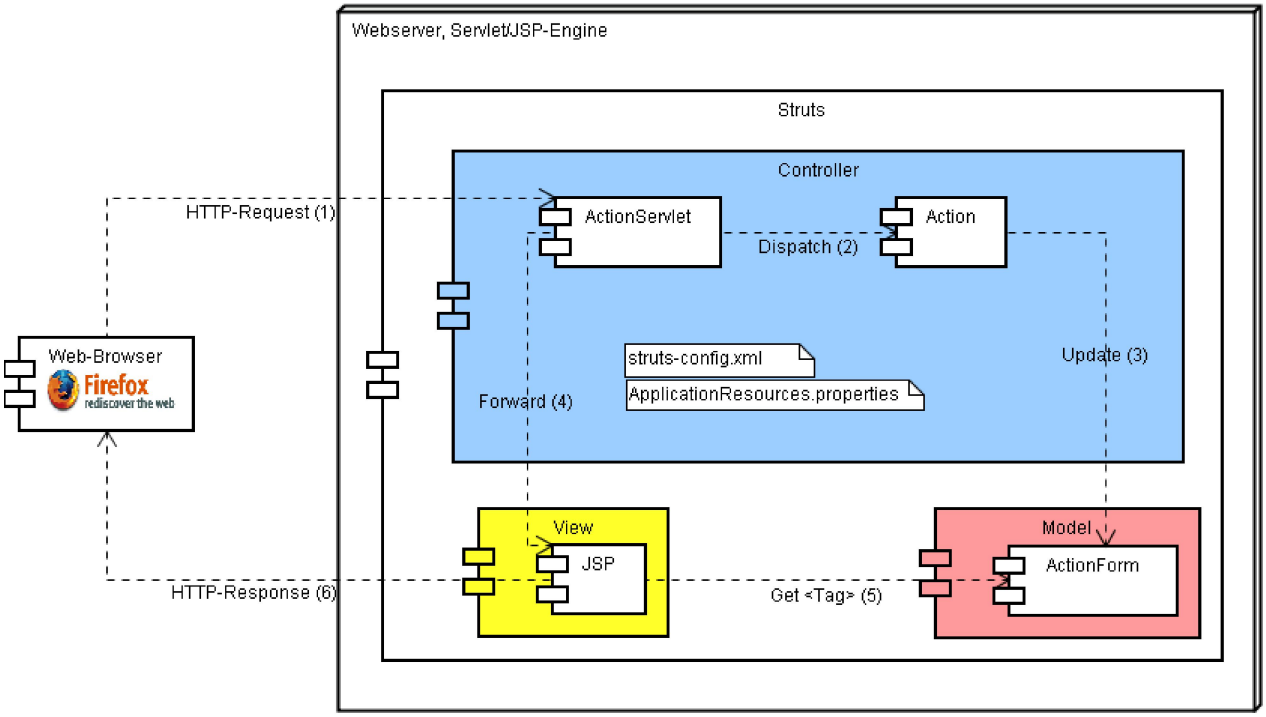

Abb. 11:Struts-Komponenten und ihr Zusammenspiel

3.4.1 Controller

Das *ActionServlet* und die *Action* bilden zusammen die Controller-Komponente. Diese beiden Klassen steuern die Abläufe der Anwendung mit klar definierten Aktivitäten. Während sich das *ActionServlet* um das Mapping der Requests zu den spezifizierten *Actions* kümmert, steuert die *Action* die Abläufe der Applikation und dient als Wrapper der Geschäftslogik.

- ActionServlet

Das org.apache.struts.action.ActionServlet ist der wichtigste Baustein des Controllers, der die HTTP-Requests empfängt, umwandelt und diese dann zur jeweiligen Action-Klasse schickt. Das *ActionServlet* benutzt dabei die *struts-config.xml*-Konfigurationsdatei in der das Mapping zwischen HTTP-Requests und den Action-Objekten definiert ist.

- Action

Die Action-Klasse ist ein Bestandteil des Controllers und eine Wrapper-Klasse der

Geschäftslogik. Die Action-Klasse soll den Workflow und die Fehlerbehandlung der Applikation kontrollieren, aber keine Geschäftslogik beinhalten. Die Geschäftslogik der Anwendung soll in anderen Komponenten (Business-Komponenten, z.B. EJBs) ausgeführt werden. Dadurch erreicht man eine saubere Trennung zwischen Präsentations-, Business- und Persistenzschicht, so dass eine Business-Komponente mit beliebigen Präsentations-Komponenten verwendet werden kann. Man muss beachten, dass das Struts-Framework nur in der Präsentationsschicht einzusetzen ist. Bei einfachen Anwendungen könnte man die Geschäftslogik in der Action-Klasse realisieren, wobei man die komponentenbasierte Architektur opfert. Für komplexe Anwendungen ist diese Trennung unbedingt notwendig. Damit das *ActionServlet* die Action-Klasse ausführen kann, muss die Action-Klasse in der *struts-config.xml*-Konfigurationsdatei angegeben werden.[14]

3.4.2 Model

Das Model in Struts ist eine von der abstrakten Klasse *org.apache.struts.action.ActionForm* abgeleitete Klasse. Sie repräsentiert den Zustand der jeweiligen Views in einer Session oder einem Request und verwaltet die Daten der Views. Die von der abstrakten Klasse *ActionForm* abgeleiteten Klassen beinhalten in einfachster Form Getter- und Setter-Methoden, jedoch keine Geschäftslogik. In dieser Klasse sind zwei wichtige Methoden implementiert, die bei jedem Request (je nach Konfiguration) vom *ActionServlet* aufgerufen werden. Mit der Methode *validate()* wird die Gültigkeit der Benutzereingabe in der HTML-Seite geprüft. Wenn das Attribut *validate* in der *struts-config.xml*-Datei auf *true* gesetzt ist, wird die Methode *validate()* vom *ActionServlet* automatisch vor dem Setzen der gemappten Felder in der Klasse *ActionForm* aufgerufen, um die Plausibilitätsprüfungen auszuführen. Die Methode *validate()* gibt ein Objekt der Klasse *ActionErrors* zurück, welchem als Container die Objekte der Klasse *ActionError* hinzugefügt werden, falls ein Fehler aufgetreten ist. Diese Fehler werden von Struts im View (JSP) an der Stelle angezeigt, an welcher der Tag `<html:errors/>` definiert ist.[14]

14 vgl. Javamagazin 04.02, Software&Support, 2002

Dieser Mechanismus ist äußerst komfortabel und bietet ein hohes Maß an Sicherheit vor unerwünschten und gefährlichen Benutzerdaten. Es kann aus einer Fülle von vorgefertigten Prüfungen, wie z.B. ob ein Eingabefeld *required* ist oder nicht, oder ob das Datum eines Feldes einer gewissen Signatur (z.B. E-Mail-Adresse) entspricht, zurückgegriffen werden. Diese in der Datei *validator-rules.xml* beheimateten Prüfungen können Dank ihrer in Java-Script umgesetzten Logik erweitert werden bzw. es können neue, eigene Regeln hinzugefügt werden. Besonders schön ist die absolut zentrale Haltung aller Regeln in der erwähnten *validator-rules.xml*, was eine Verzettelung von Prüfungslogik auf einzelne JSPs verhindert und somit über den dadurch entstandenen Komfort direkt zu mehr Sicherheit führt.

3.4.3 View

Der View im Struts ist eine einfache JSP-Datei, welche die HTML- und Struts-Tags beinhaltet. Struts befreit die JSP-Seiten von Java-Code durch die eigenen Tags. Alle HTML-Tags sind mit Struts-eigenen Tags abgebildet. Die JSP-Seiten beinhalten dabei weder Workflow noch Geschäftslogik. So benutzt Struts verschiedene Tags zum Zugriff auf Daten des Models,

z.B. `<html:text property="username"/>`.

Man benötigt dadurch in Struts keinen Java-Code in der JSP. Struts übernimmt diese Arbeit und ersetzt den Key `username` mit dem passenden Value aus der für diese ActionForm gültigen Datei *Resources.properties*. Die JSP braucht nur zu wissen, wie die gültige Property-Datei der ActionForm heißt. Struts bindet diese Properties der ActionForms mit den HTML-Komponenten, um die Values der Properties in HTML anzuzeigen.[15]

Im Hinblick auf Internationalisierung ist dieser Mechanismus besonders komfortabel. Je nach gewünschter Sprache wird eine andere Properties-Datei geladen und `username` durch „Username“, „Benutzername“ oder Bezeichnungen aus weiteren Sprachen ersetzt. Es sei noch einmal erwähnt, dass das Herzstück einer Struts-Applikation die Konfigurationsdatei *struts-config.xml* ist. In ihr werden die MVC-Komponenten verwoben und die ganze Applikation deklarativ konfiguriert.

15 vgl. Javamagazin 04.02, Software&Support, 2002

3.5 Java Management Extention, JMX

Heutige serverseitige Java-Applikationen sollten leicht und komfortabel verwaltbar sein. Hierfür gab es in der Vergangenheit keine einheitliche Lösung, jeder erschuf sich eigene Management-Schnittstellen zu seiner Applikation. Mit der JMX Spezifikation trat SUN an, diese Lücke zu schließen und eine einheitliche Schnittstelle für das Management von Java Anwendungen zu liefern.[16]

Bei JMX handelt es sich um eine Spezifikation, die Architektur, Design-Patterns, APIs sowie Basisdienste für Anwendungs-, Netzwerk- und Geräte-Administration in der Java-Programmiersprache beschreibt. Damit wird ein Framework zur Verfügung gestellt, mit dem Entwickler auf einfache Weise Managementfunktionalitäten in Java implementieren und in ihren Anwendungen integrieren können. Die JMX Spezifikation ist im Rahmen des Java Community Process unter Beteiligung großer Firmen wie z. B. IBM, BEA Systems und Borland entstanden. Auch die Apache Software Foundation ist Teil der Expert Group, welche die JMX Spezifikation entwickelt. Viele Serveranwendungen setzen mittlerweile auf die JMX Spezifikation. So bringen z.B. JBoss, Tomcat aber auch BEAs Weblogic oder IBMs Websphere bereits von Haus aus JMX Schnittstellen in Form der Implementierten JMX Spezifikation mit. Die Ansteuerung einer durch JMX verwaltbaren Ressource, wie z.B. eine Anwendung oder ein Anwendungsteil, wird durch Managed Beans (MBeans) realisiert, die eine zentrale Rolle in der JMX-Architektur spielen.[17]

3.5.1 MBeans

Jede MBean kapselt eine Ressource (Anwendung, Netzwerk,...) und stellt eine Schnittstelle (ein so genanntes Management Interface) zur Verfügung, über die einerseits der Zustand der Ressource abgefragt und verändert werden kann, und andererseits sich bestimmte Operationen auf die Ressource ausführen lassen. Weiter wurde für MBeans auch ein Notifikationsmechanismus eingeführt, der das Erzeugen

16 vgl. http://www.oio.de/public/java/jmx/jmx.html, 30.01.05
17 vgl. Javamagazin 06.02, Software&Support, 2002

und Propagieren von Ereignissen ermöglicht. Das Management Interface einer MBean besteht genauer aus:

- Attributen, vergleichbar den Properties von JavaBeans
- Operationen
- Notifikationen
- Public-Konstruktoren der MBean-Klasse

JMX hat vier Typen von MBeans definiert:
Standard MBeans, Dynamic MBeans, Open MBeans und Model MBeans.
In der Konfiguration der nM-Plattform werden ausschließlich Standard MBeans verwendet. Dies ist die am schnellsten zu erlernende und am einfachsten einzusetzende Art von MBean.[18]

3.5.2 JMX-Agent

Das Produkt, welches die JMX-Spezifikation implementiert, implementiert auch einen sog. JMX-Agenten, dessen zentrale Komponente der MBean-Server darstellt. Der JMX-Agent stellt eine Brücke zwischen den MBeans und den Managementapplikationen dar, so dass die MBeans von den Managementapplikationen angesteuert werden können. Für die Anbindungen von verschiedenen Zugriffsmöglichkeiten und Protokollen (HTTP, SNMP, RMI) an JMX-Agenten sind Protokolladaptoren und Konnektoren vorgesehen. Darüber hinaus werden Basisdienste wie Monitoring, Timer usw. vom Agent angeboten.
Der eingesetzte JBoss Application Server besitzt eine Implementierung des *HTTP-Protocol-Adapter*, welcher es erlaubt per Standard-Webbrowser den Agenten zu kontaktieren und somit auf die im MBean-Server registrierten MBeans zuzugreifen. Serviert wird der Webbrowser durch die *JMX Console Web Application* des JBoss, eine webbasierte, serverseitige Management-Applikation.[19]

18 vgl. http://docs.jboss.org/admin-devel/AdminDevelTOC.html, 30.01.05
19 vgl. Javamagazin 06.02, Software&Support, 2002

4 Eingesetzte Design Patterns

4.1 Session-Fassade

In einer mehrschichtigen J2EE-Anwendungsumgebung treten u.a. Probleme durch eine zu enge Kopplung von Komponenten auf, die zu direkter Abhängigkeit zwischen einzelnen Komponenten führen. Weiter ist das Fehlen einer einheitlichen Zugriffsstrategie für Komponenten untereinander problematisch. An die Anwendung bestehen Anforderungen in der Abschirmung von Komponenten untereinander und dem Erhalt ihrer Unabhängigkeit. Weiter ist die Vereinfachung von Business-Schnittstellen gefordert[20]. Im Fall der nM-Plattform ist vor allem die Entkopplung von Komponenten und der Aspekt der Zugriffsstrategie wichtig, um dem gewünschten modularen Aufbau und dem Plattform-Charakter der Applikation gerecht zu werden. Mit dem Einsatz des Session-Fassade-Patterns kann geholfen werden, diese Grundsätze umzusetzen. Dazu ist das Pattern zu folgenden Leistungen fähig:

- Bereitstellen einfacher Schnittstellen für Komponenten, indem alle komplexen Interaktionen zwischen anderen Komponenten verborgen werden.
- Komponenten, welche eine Session-Fassade nutzen, werden die zugrunde liegenden Interaktionen und gegenseitigen Abhängigkeiten zwischen anderen Komponenten verborgen. Dadurch erreicht man eine bessere Verwaltbarkeit, Zentralisierung der Interaktionen, größere Flexibilität und bessere Möglichkeiten, mit Änderungen fertig zu werden.
- Bereitstellen einer grobkörnigen Dienstebene, um die Business-Implementierung von der Business-Abstraktion zu trennen.
- Vermeidung der direkten Offenlegung der zugrunde liegenden Geschäftsobjekte für bestimmte Komponenten, um eine enge Kopplung zwischen den Schichten minimal zu halten.

Die Lösung liegt in der Verwendung von Session Beans als Fassaden, um die Komplexität der Interaktionen zwischen Komponenten zu kapseln, die an einem Workflow beteiligt sind.

20 vgl. Adam Bien, Entwurfsmuster für die J2EE, Addison-Wesley, 2002

Die Session-Fassade verwaltet die Geschäftsobjekte und bietet anderen Komponenten eine einheitliche, grobkörnige Dienstzugriffsschicht.[21]

4.2 Value Object

Das *Value Object Pattern* ermöglicht das Zusammenfassen von mehreren Parametern zu einem Objekt. Es handelt sich bei einem Value Object um einen „Datenhalter“, der dazu benutzt wird, Methodensignaturen zu vereinfachen. Zu diesem Zweck werden die Informationen, die nötig sind um mehrere Methoden aufzurufen, in einem Objekt zusammengefasst und auf einmal übergeben. Ein Value Object besteht aus einem Konstruktor und einer Reihe von get-Methoden, die den Zugriff auf die gehaltenen Daten ermöglichen. In verteilten Umgebungen ist bei Verwendung von Value Objects das Laufzeitverhalten besser, als bei der seriellen Übertragung jedes einzelnen Parameters. Es ist auch der besseren Wartbarkeit der Applikation dienlich. Allerdings ist zu beachten, dass bei falscher Benutzung künstliche Holder-Strukturen entstehen können, die nur dazu dienen, nicht zusammengehörige Parameter zusammenzufassen.[22]

21 vgl. Adam Bien, Entwurfsmuster für die J2EE, Addison-Wesley, 2002
22 vgl. Adam Bien, Enterprise Java Frameworks, Addison-Wesley, 2001

5 Architektur

In Fortführung der Diskussion der Aufgabenbereiche und der allgemeineren Architekurbetrachtungen der nM-Plattform wird eine Paketstruktur erstellt, welche den modularen und ebenenbezogenen Aufbau der nM-Plattform widerspiegelt.
Die Namen der Pakete sind dabei so gewählt, dass intuitiv klar sein soll, welche Aufgaben die darin enthaltenen Klassen haben. Weiter wird die nM-Plattform anhand des J2EE-Mehrschichtmodells erläutert. Damit sollen die Komponenten und deren Aufgaben und Funktion noch klarer werden.

Es existieren zwei große Paketfamilien: Kern-Pakete und Service-Pakete (Abb. 12). Das Paket `de.netads.kern`, seine Unterpakete und die darin enthaltenen Klassen bilden das Fundament der nM-Plattform-Applikation, auf das der SMSmonkey-Service und zukünftige Services aufsetzen. Im Paket `de.netads.smsmonkeyservice` und seinen Unterpaketen befinden sich die Klassen des SMSmonkey-Services.

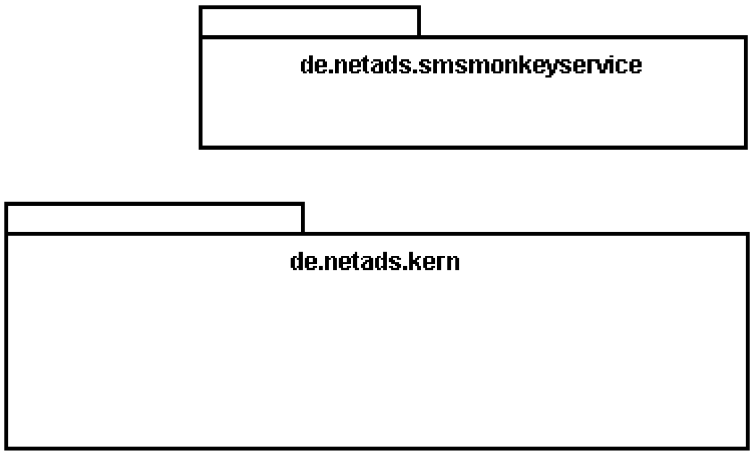

Abb. 12: Kern-und Servicehauptpakete

5.1 Kernpakete

Zur Familie der Kernpakete gehören die Unterpakete des Pakets `de.netads.kern`:

- `verwaltung`
- `nachrichtenversand`
- `web`
- `filter`
- `hilfe`
- `konfiguration`
- `datenobjekte`
- `persistenz`

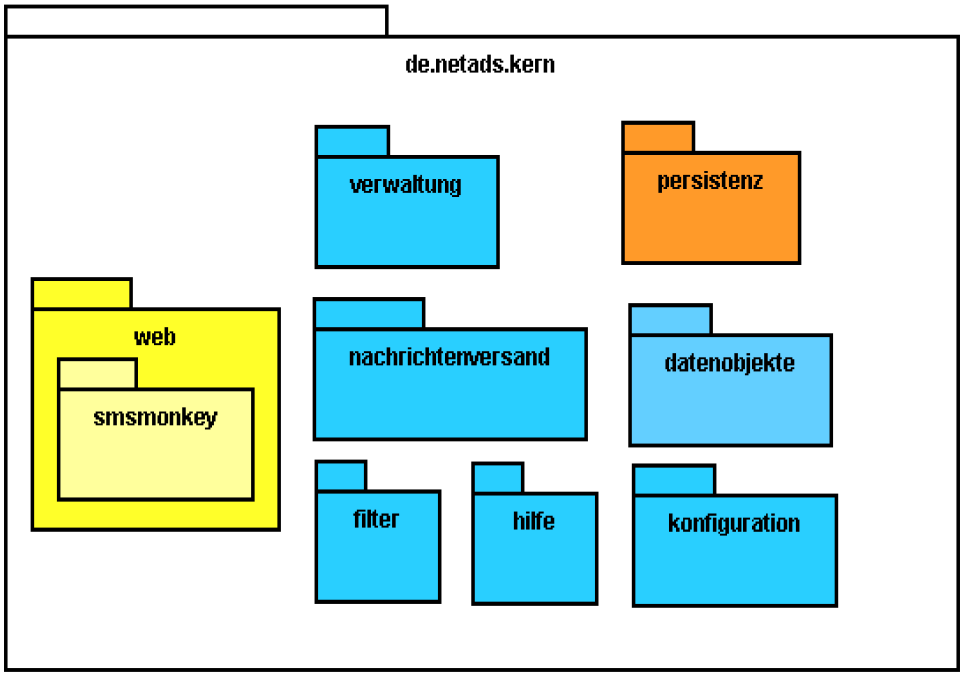

Abb. 13: Übersicht Paket Kern *und seine Unterpakete*

- de.netads.kern.verwaltung

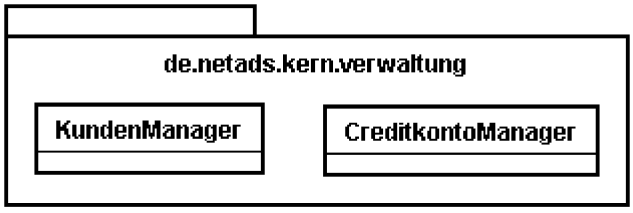

Abb. 14: Verwaltung

In diesem Paket befinden sich Klassen (Stateless Session Beans), welche für die

Verwaltung von Kunden- und Creditkonto-Daten verantwortlich sind. Es sind Schnittstellen für die speziellen, Service-eigenen Verwaltungsklassen und für die Klassen des Struts-Web-Frameworks im Paket `de.netads.kern.web`, welche den webbasierten Anteil der nM-Plattform leisten. Die Klassen der Verwaltung stehen als Fassade vor den Entity Beans der Datenhaltung. Sie kapseln und verstecken deren Datenhaltungslogik. Zum Beispiel werden hier Datenobjekte befüllt und dabei Typumwandlungen vorgenommen, welche die Nutzung der Value Objects angenehmer machen.

- de.netads.kern.nachrichtenversand

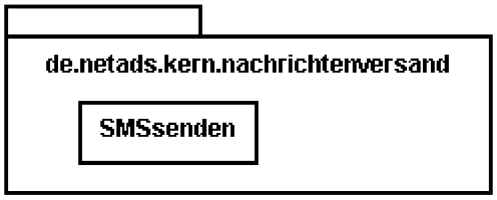

Abb. 15: Nachrichtenversand

Die Session Beans im Paket `nachrichtenversand` leisten den Versand von Nachrichten. Diese Funktionalität kann von allen Services genutzt werden. Ihre Methoden dienen als Schnittstellen für Services und Struts-Klassen und verstecken die dahinter stehende Zusammenarbeit mit Verwaltung und Datenhaltung. Als Beispiel sei das Session Bean `SMSsenden` genannt, welches den Versand von SMS anbietet und anhand Methoden verschiedenster Signaturen zur einfachen Benutzung bereitsteht. Im Zuge einer Weiterentwicklung der nM-Plattform können genau dort Klassen angesiedelt werden, welche z.B. den Versand von E-Mails zur Aufgabe haben.

- de.netads.kern.web

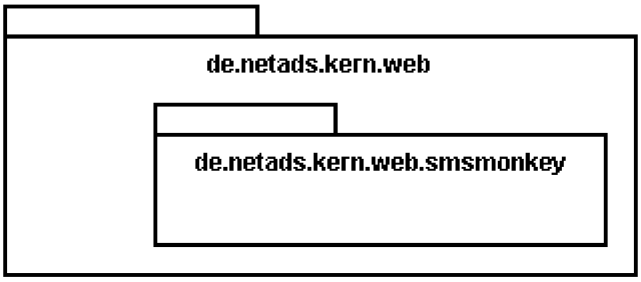

Abb. 16: Heimat der Struts-Klassen

Im Paket `de.netads.kern.web` befinden sich die Klassen des Struts-Web-Frameworks, mit dessen Hilfe alle webbasierten Belange der nM-Plattform behandelt werden. Das Paket teilt sich weiter in je ein Paket für jeden Service auf, in dem dann z.B. Action- und Form-Klassen des entsprechenden Services untergebracht sind.
Damit soll erreicht werden, dass die einzelnen Action- und Form-Klassen, sowie Tag-Bibliotheken nicht verstreut, sondern zentral zusammengehalten sind.
In diesem Punkt steht das Paket `de.netads.kern.web` der Philosophie der Trennung von Kern- und Service-Angelegenheiten auf verschiedenen Ebenen der nM-Plattform entgegen, begründet sich aber durch die Komplexität von Struts. Bei einer horizontalen Verteilung der Struts-Klassen, d.h. bei der Verteilung über mehrere Pakete in mehreren Ebenen, kann es schwierig werden den Überblick über den Web-Applikations-Anteil zu bewahren, was dem übergeordneten, größeren Ziel, eine gute Wartbarkeit der nM-Plattform zu gewährleisten, nicht zuträglich ist.

- de.netads.kern.filter

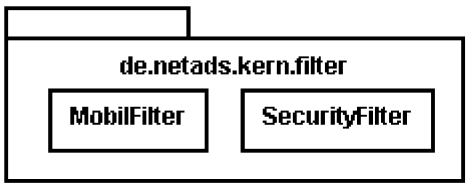

Abb. 17: Filtermechanismen

Wie der Name des Pakets andeutet beinhaltet es Filterklassen, welche Nutzerdaten auf z.B. Datentypen-Konformität prüft. Das Spendieren eines eigenen Pakets alleine

für Servletfilter und anderer Filterklassen soll der Erweiterbarkeit und Sicherheit der nM-Plattform zuträglich sein. Es gilt das gleiche wie für das Paket *web*. Eine Verteilung von Filtern auf die Services könnte zu Redundanzen und Fehlern führen. Die Aufgaben der Filter sind viel zu wichtig und zu sicherheitsrelevant, als dass man sich dies erlauben könnte. Weiter ist es mit der zentralen Unterbringung aller Filter-Belange für alle Services und den Kern der nM-Plattform möglich, ein wiederverwendbares Filter- und Sicherheits-Framework innerhalb der nM-Plattform aufzubauen.

- de.netads.kern.hilfe

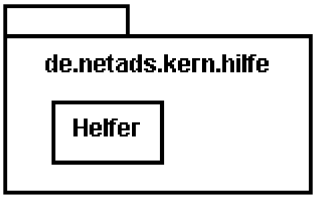

Abb. 18: Unterstützende Hilfe

In diesem Paket befindet sich eine Session Bean, welches als Unterstützer der anderen Beans gesehen werden soll. Es liefert und verarbeitet Daten allgemeiner Natur, die sich schlecht einem anderen Geschäftsbereich zuschlagen lassen. Zum Beispiel besitzt es Methoden zum Generieren von Hash-Werten von Zeichenketten, die von allen Bereichen des Kerns und der Services genutzt werden können.
Dieses Paket ist Ausdruck dafür, dass sich nicht alle Angelegenheiten in einer Applikation benennen und in eine Struktur pressen lassen, bzw. trotz Struktur dennoch kein geeigneter Platz für manche Aufgabe existiert.
Das Paket soll Heimat für Beans und Standard-Java-Klassen mit Zuarbeiter-Charakter sein, welche mit ihren Methoden Schnittstellen für Kern und Services liefern. Zukünftige Entwickler der nM-Plattform müssen aber darauf achten, dass dieses Paket nicht verwildert, da sonst die Struktur der ganzen Applikation darunter leidet.

- de.netads.kern.konfiguration

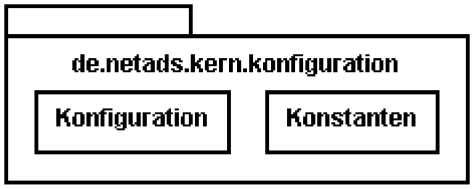

Abb. 19: Verarbeitung von Konfigurationsdaten

Im Paket `de.netads.kern.konfiguration` sind alle Klassen beheimatet, die in Verbindung mit dem JMX-Management-Bereich der nM-Plattform stehen. Als Unterpaket von `de.netads.kern` bezieht sich die Verantwortlichkeit dieser Klassen auf Konfigurationsangelegenheiten der Kern-Ebene der nM-Plattform. Die Session Beans dieses Pakets dienen als Bindeglied zwischen JMX und dem Kern der nM-Plattform. Die sichtbaren Methoden dieser Klassen sind Anlaufpunkte für den Web-Applikations-Teil, sowie Business und Verwaltung der Kern-Ebene. Die Service-Ebenen haben hier die Möglichkeit feste oder variable Werte der Kern-Ebene zu beziehen.

- de.netads.kern.datenobjekte

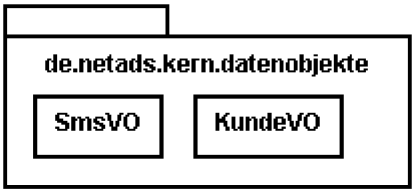

Abb. 20: Datenobjekte

Im Paket `de.netads.kern.datenobjekte` siedeln alle Datenobjekte der Kern-Ebene. Um die Klassen nicht in den Paketen der persistenten Datenhaltung untergehen zu lassen, ist ein eigenes Paket für sie vorgesehen. Da sich darin nicht nur dem Pattern entsprechende Value Objects, sondern auch einfache Java-Klassen als Datencontainer befinden können, wird dadurch eine Bündelung und Zentralisierung aller Datenobjekte des Kerns erreicht, was der Wartbarkeit und

Übersichtlichkeit der Applikation zu Gute kommen soll.

- de.netads.kern.persistenz

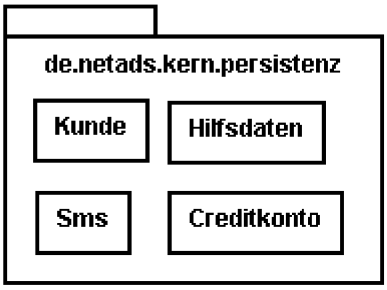

Abb. 21: CMP Entity Beans

In diesem Paket sind alle CMP Entity Beans hinterlegt, welche die persistente Datenhaltung der Geschäftsdaten der Kern-Ebene zur Aufgabe haben. Die Klassen dieses Pakets stehen in enger Zusammenarbeit mit den Klassen des Pakets der Datenobjekte.

Die Pakete der nM-Plattform geben somit einen Ort in der Applikation vor, wo sich Klassen, welche dem entsprechende Aufgaben zu erfüllen haben, anzusiedeln sind. Die Struktur und Benennung der Pakete setzt sich auf Service-Ebene fort. Die Klassen in den Paketen der Service-Ebenen nehmen die gleichen Aufgaben wahr, wie ihre Namenskollegen in den Kern-Paketen. Der Unterschied besteht darin, dass sie dies in einer Service-spezifischen Weise tun und nur für den jeweiligen Service verantwortlich sind. Oft führen sie auch nur eine Umsetzung der Daten aus den Schnittstellen des Kerns durch.

5.2 Servicepakete

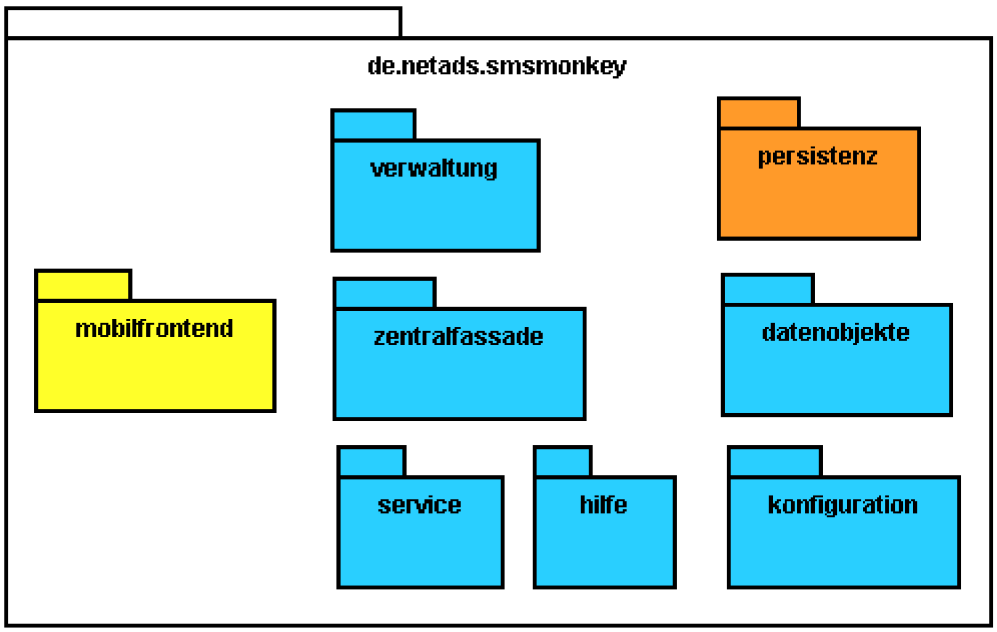

Abb. 22: Übersicht Paket smsmonkey *und Unterpakete*

Die Unterpakete des Paketes `de.netads.smsmonkeyservice` sind:

- `verwaltung`
- `zentralfassade`
- `mobilfrontend`
- `service`
- `hilfe`
- `konfiguration`
- `datenobjekte`
- `persistenz`

Die meisten Pakete sind gleich benannt, wie Pakete des Kerns. Die in Paketen mit gleichem Namen enthaltenen Klassen erfüllen die gleichen Aufgaben wie ihre Namensvetter in den Kernpaketen, nur sind die Aufgaben Service-bezogen und nicht allgemein. Aus dem Rahmen fallen die Pakete `zentralfassade`, `mobilfrontend` und `service`, in denen die Klassen rein Service-spezifische Geschäftslogik umsetzten und keinen Namensvetter in den Kern-Paketen haben. Im Folgenden soll auf den SMSmonkey-Service eingegangen werden. Dieser dient als Referenz-Service für zukünftige Services der nM-Plattform. Ziel ist es, in einem zukünftigen Service die Pakete und Klassen des SMSmonkey-Services nachzubauen, um nur noch die zentrale Geschäftslogik des Services implementieren zu müssen.

- de.netads.smsmonkeyservice.zentralfassade

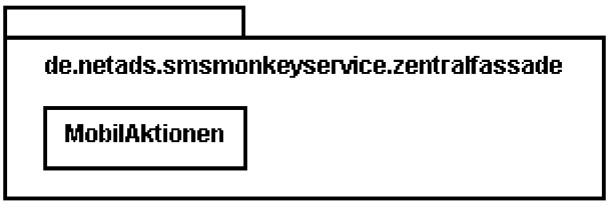

Abb. 23: Anlaufpunkt für mobile Clients

Wie der Paketname schon ausdrückt, befindet sich an dieser Stelle eine Fassade mit zentraler Bedeutung. Im Falle des SMSmonkey-Services stellt diese das Stateless Session Bean `MobilAktionen` dar. Diese Klasse dient als Bindeglied zwischen den Belangen der mobilen Clients und der Geschäftslogik der Services, bzw. des Kerns. Sie ist der einzige Anlaufpunkt für Komponenten wie Servlets oder Servletfilter, welche in direkter Kommunikation mit den Clients stehen. Sie bietet alle nötigen Schnittstellen zur Geschäftslogik der Serviceleistung, Verwaltung und Datenhaltung und implementiert selbst noch einen kleinen Teil vorverarbeitender Logik.

- de.netads.smsmonkeyservice.mobilfrontend

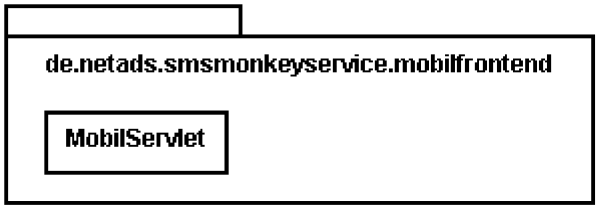

Abb. 24: Partner der mobilen Clients

In diesem Paket geschieht die Kommunikation mit den mobilen Clients. Alle Komponenten, wie das `MobilServlet` des SMSmonkey-Services, die in direkter Kommunikation mit einem mobilen Client stehen, haben hier ihre Heimat. Ausgenommen davon sind allerdings filternde Klassen, wie z.B. Servletfilter, welche ihr eigenes Paket haben.
Es werden hier Client-Daten verarbeitet, welche eventuell schon durch einen Filtermechanismus gelaufen sind, wie es beim SMSmonkey-Service der Fall ist. Der Kommunikationspartner der Klasse `MobilServlet` und alleiniger Zugang zur

Geschäftslogik ist das besprochene MobilAktionen-Bean im Paket `zentralfassade`.

- de.netads.smsmonkeyservice.service

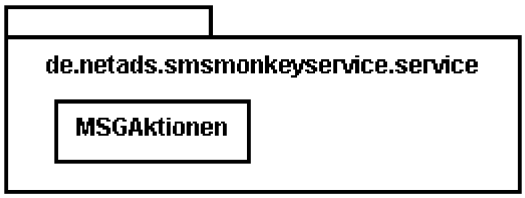

Abb. 25: Business-Logik eines Services

Im Paket *service* sitzt die eigentliche Geschäftslogik eines Services. In diesem Fall die des SMSmonkey-Services.
Da der Versand von SMS als allgemeines, von allen Ebenen (Services und Kern) nutzbares Kommunikationsmittel eingestuft ist und sich somit auf der Kern-Ebene der nM-Plattform angesiedelt ist, befindet sich an dieser Stelle die Geschäftslogik des MSG-Versands, die hoch spezielle Serviceleistung des SMSmonkey-Services.
Die Klasse `MSGAktionen` ist über die Zentralfassade Anlaufpunkt für mobile Clients, sowie direkt für Struts-Action-Klassen aus dem Web-Bereich und bedient sich ihrerseits der Leistungen von Kunden- und Kontoverwaltung.

- de.netads.smsmonkeyservice.verwaltung

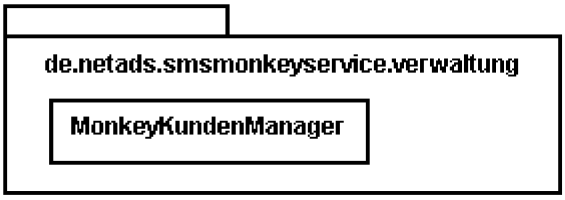

Abb. 26: Verwaltung des Services

Dieses Paket enthält, wie das gleichnamige Paket des Kerns auch, Klassen zur Verwaltung, aber speziell von Service-spezifischen Kunden- und Kontodaten. Der SMSmonkey-Service benötigt eine Umsetzung von generellen Kundendaten aus der Kern-Ebene zu Service-spezifischen Kundendaten, was die Klasse

`MonkeyKundenManager` leistet. Sie stellt eine Fassade der Service-bezogenen Kundendaten-Verarbeitung dar und versteckt den damit verbundenen Aufwand vor der Zentralfassade und den Struts-Klassen.

- de.netads.smsmonkeyservice.datenobjekte

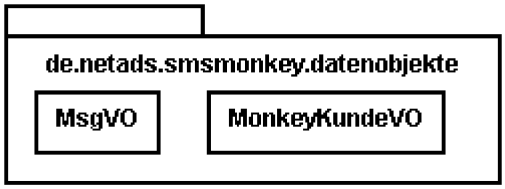

Abb. 27: Datenobjekte

Wie in der Familie der Kern-Pakete, hat das Paket `datenobjekte` des SMSmonkey-Services die Aufgabe Datenobjekte eine Heimat zu geben. Als Unterpaket von `smsmonkeyservice` betrifft dies nur Service-spezifische Datenobjekte.

- de.netads.smsmonkeyservice.persistenz

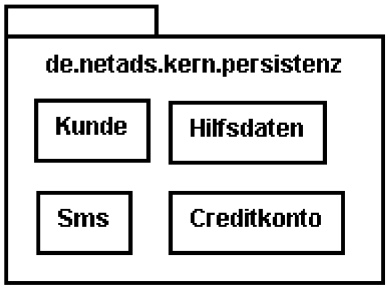

Abb. 28: Beans der Datenhaltung

Wie in den Kern-Paketen auch, siedeln in diesem Paket die Entity Beans der Datenhaltung. Sie leisten die Vorhaltung Service-spezifischer Daten des SMSmonkey-Services, welche von den jeweiligen Schnittstellen, z.B. den SMSmonkey-Verwaltungsklassen, bezogen und gesetzt werden können.

- de.netads.smsmonkeyservice.hilfe

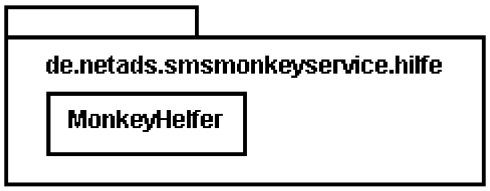

Abb. 29: Unterstützende Klassen

Das Paket `de.netads.smsmonkeyservice.hilfe` ist Heimat von Klassen mit Steigbügelfunktion, speziell für die Logik des SMSmonkey-Services.

- de.netads.smsmonkeyservice.konfiguration

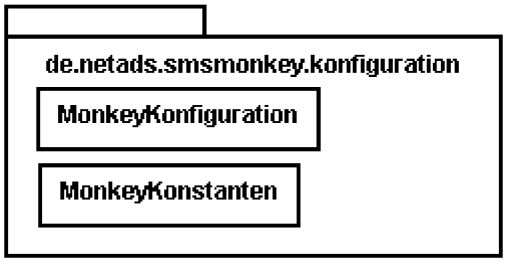

Abb. 30: Verarbeitung v. Konfigurationsdaten

Die Paketstruktur des Kerns findet sich auch in diesem Paket wieder. Die in diesem Paket enthaltenen Klassen dienen der Kommunikation mit dem JMX-Bereich und der Verarbeitung daraus bezogener, variabler Werte. Weiter befindet sich hier die Haltung von programmativ festgeschriebenen, Service-spezifischen Konstanten des SMSmonkey-Services.

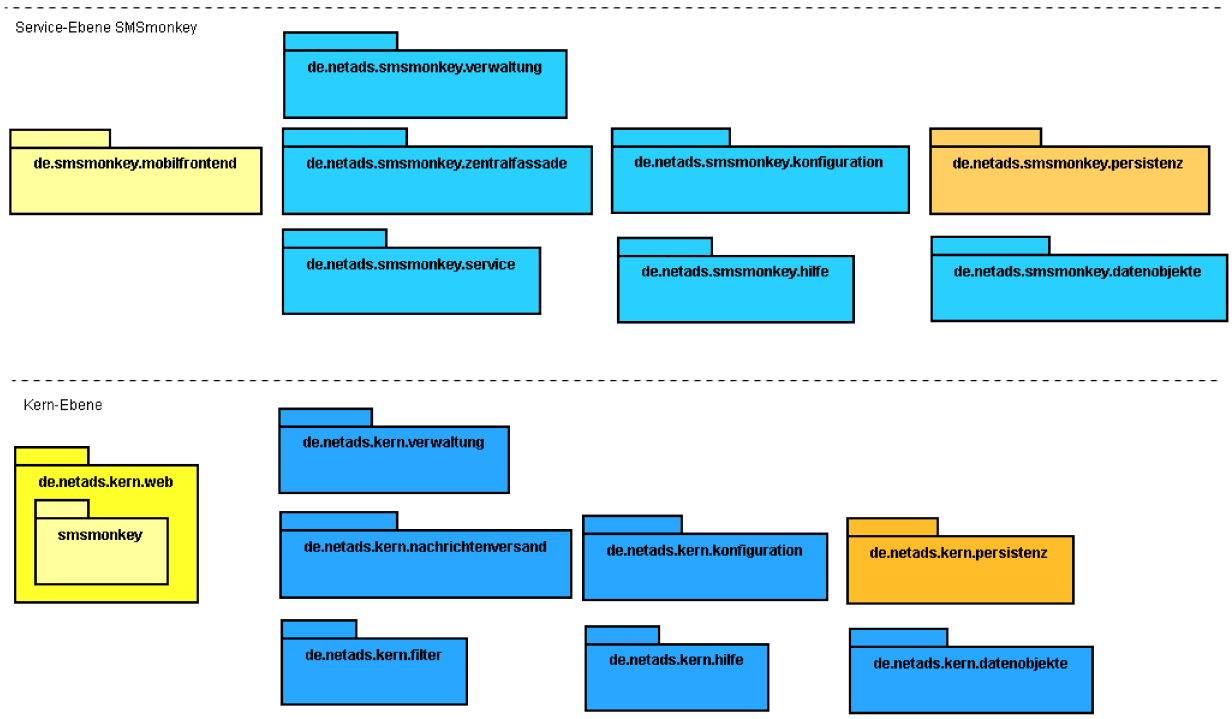

Abb. 31: Übersicht der Kern- und Servicepakete

Die Paketstruktur des SMSmonkey-Services ist eine Referenz für neu zu implementierende Services. Das Wissen um die Struktur und die Aufgaben der darin befindlichen Klassen soll die Umsetzung eines neuen Services vereinfachen.
Um einen gesamten Überblick über die Applikation zu bekommen und den Brückenschlag zur Analyse, in denen die Aufgabenbereiche und die Ebenenverteilung besprochen wurde, zu schaffen, sei Abbildung 31 betrachtet. Die Pakete und die darin enthaltenen Klassen erfüllen entsprechend ihrer Ebene und ihrem Bereich Präsentations-, Geschäfts- oder Datenhaltungsaufgaben

5.3 Mehrschicht-Architektur

Die nM-Plattform ist eine mehrschichtige (multi-Tier) Applikation (Abb. 32). Im Folgenden soll die Struktur der Applikation anhand des Schichtmodells der J2EE-Spezifikation beschrieben werden. Dies soll die Architektur abrunden und zu einem klareren Bild der nM-Plattform führen. Die auftauchenden Widersprüche zwischen dem theoretischen Modell der Tiers und der „lebendigen“, praktischen Applikation sollen zur Reflexion anregen und dem Gesamtverständnis zuträglich sein.

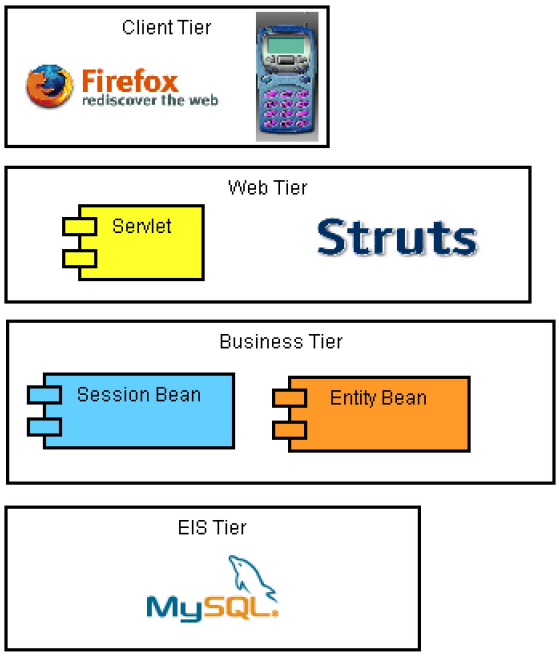

Abb. 32: nM-Plattform Schichtenmodell

Im Kontext einer J2EE-Architektur werden vier Schichten (engl. Tier) unterschieden.[23]
In der nM-Plattform existiert die Kern-Ebene und die Service-Ebenen. Jede Ebene entspricht für sich dem Mehrschichtmodell und ist relativ lose mit anderen Ebenen verwoben. Es kann also gesagt werden, dass die nM-Plattform eine Applikation ist, die in sich mehrere multi-Tier-Applikationen beheimatet und somit selbst zur (schwach verteilten) Mehrschichtapplikation wird. Eine stärkere Verteilung einzelner Ebenen auf verschiedene Maschinen ist anhand dessen, bei Bedarf und je nach Wachstum der nM-Plattform, gezielt und leicht zu erreichen.

5.3.1 Client Tier

Der Client-Tier dient der Interaktion mit dem Nutzer, er stellt dem Nutzer Informationen bereit, die ihm vom System zur Verfügung gestellt werden. Im Kontext der J2EE-Architektur werden verschiedene Clients unterschieden, u.a. HTML Clients und Java Applikationen.[23]

23 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

Bei der nM-Plattform zählen hierzu die mobilen Clients der Services z.B. MIDlets oder Symbian-OS-Anwendungen und Webbrowser für das webseitige Angebot.

5.3.2 Web Tier

Der Web Tier verwaltet Darstellungslogik, die für die Aufbereitung und die Darstellung von Informationen im Client Tier benötigt wird. Darüber hinaus nimmt dieser Tier die Informationen von Nutzern entgegen, die diese dem Client Tier übermitteln. Der Web Tier generiert Antworten auf die Nutzereingaben und übermittelt diese an den zugrunde liegenden Business Tier oder von diesem zurück an den Client Tier. Im Kontext der J2EE-Architektur wird die Darstellungslogik in Gestalt von Servlets und von JavaServer Pages in Webcontainern realisiert [24].
Konkret auf die nM-Plattform bezogen stehen den mobilen Clients der Services Servlets im Web Tier gegenüber (z.B. die Klasse `MobilServlet`) und alle webbasierten Präsentationsbelange werden anhand der logisch in der Web Tier angestammten Klassen des Struts-Web-Frameworks behandelt. Dazu gehören die Klassen im Paket `de.netads.kern.web` und dessen Unterpakete.

5.3.3 Business Tier

Der Business Tier verwaltet die eigentliche Geschäftslogik des Systems. Komponenten der Business Tier, die in Gestalt von Enterprise JavaBeans (EJBs) realisiert sind, führen Geschäfts-Aktivitäten aus oder aktivieren andere Komponenten. EJBs dienen der Durchführung der vom Nutzer oder vom System angestoßenen Aktivitäten zur Verfolgung eines Geschäftszieles. EJBs regeln dabei auch implizit ihren Lifecycle, indem sie auf ein Transaktionshandling, die Mechanismen zur Persistierung und auf sichere Zugriffsmechanismen zurückgreifen können; dabei werden EJBs in einem sog. EJB-Container abgelegt, der diese Dienste für die EJB-Komponente bereithält.[24]
In der nM-Plattform fallen darunter alle Klassen, die nicht zur Web Tier zählen. Die (persistenten) Entity Beans zählen ebenso hierzu, wie alle mit Geschäftsaufgaben

24 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

betrauten Session Beans, sowie standard Java-Klassen, wie z.B. Datenobjekte.

5.3.4 EIS Tier

Dieser Tier ist für die Anbindung von EIS-Systemen (Enterprise Information Systemen) zuständig. Diese Systeme sind u.a. Datenbank-Systeme.[25] Auf den EIS-Tier soll nicht weiter eingegangen werden, da er unter Verwendung moderner J2EE-Server und J2EE-Technologien (z.B. CMP 2.0) deklarativ austauschbar und transparent ist.

Ein besonderer Teil der nM-Plattform ist die Konfiguration. Diese läßt sich nicht eindeutig einer bestimmten Tier zuordnen. Sie ist eine eigenständige Applikation auf dem Application Server (aber dennoch in die nM-Plattform eingewoben) und kann mehr als Client-Applikation der Business Tier gesehen werden, als fest zur Business Tier zugehörig. Ebenfalls strittig ist die eindeutige Zuordnung der Servletfilter-Klassen. Zum Einen stehen sie in direkter Verwandtschaft zu Servlets, sind nahe an der Client Tier zuhause, zum Anderen aber beinhalten sie ein großes Maß an Geschäfts- bzw. Filterlogik und könnten deshalb genauso der Business Tier zugeschlagen werden. In der nM-Plattform sind Filter der Business Tier zugehörig.

25 vgl. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004

6 Design am Beispiel

6.1 Fassaden

Nachdem in den vorhergehenden Kapiteln besprochen wurde, welche Aufgaben, in welchen Bereichen der nM-Plattform erfüllt werden und in welche architektonische Struktur die nM-Plattform gegossen ist, soll nun ausschnittsweise tiefer auf den Einsatz bestimmter Design-Patterns und auf das Zusammenspiel verschiedener Klassen eingegangen werden.

Bestimmte Klassen des Geschäfts-Bereichs der nM-Plattform folgen dem Session Fassade Pattern. Dies ermöglicht eine lose Kopplung zwischen Präsentations-, Geschäfts- und Datenhaltungsbereichen innerhalb einer Ebene und zwischen den Geschäftsbereichen verschiedener Ebenen, genauer zwischen Service-Ebenen und Kern-Ebene. Durch die Methoden der Fassaden-Klassen werden konkrete Zugriffspunkte für andere Klassen angeboten, welche die Leistungen der Fassaden nutzen wollen. Erst durch den Einsatz des Fassade-Patterns wurde die Realisierung der nM-Plattform-Applikation im Sinne einer Plattform mit darauf aufsetzenden Services erreicht. Auf Kern-Ebene sind vor allem die Klassen *KundenManager* und *CreditkontoManager* als Fassaden anzusehen, sowie auf den Service-Ebenen die Klasse *MobilAktionen.*

Die Klassen *KundenManager* und *CreditkontoManager* verstecken und kapseln das Kunden- und Kontodaten-Management und die Haltung der betreffenden Daten. Ein Benutzer dieser Fassaden ist die logisch auf Kern-Ebene angesiedelte, allgemeine Web-Infrastruktur, von wo aus sie von den Struts-Action-Klassen benutzt werden. Als Beispiel einer solchen Fassadentätigkeit sei die Methode *addKunde(..)* der KundenManager-Klasse genannt (Abb. 33). Anhand der Daten, welche die Methode von einer Struts-Action-Klasse übergeben bekommt, legt die Fassaden-Klasse einen neuen Kunden im System an. Sie benutzt dazu andere Klassen des Systems, wie die Klasse *CreditkontoManager* und unterstützende Klassen wie die Klasse *Helfer* und veranlasst die persistente Datenhaltung durch Einsatz des CMP Entity Beans *Kunde*.

Ein Stück weit erledigt die Fassaden-Klasse aber auch Geschäftslogik, in dem sie u.a. die richtigen Initialwerte für den anzulegenden Kunden heraussucht und der Datenhaltung zuführt. Dies alles geschieht für den Struts-Bereich der nM-Plattform transparent, welcher nur auf eine Erfolgs- oder Fehlermeldung wartet.

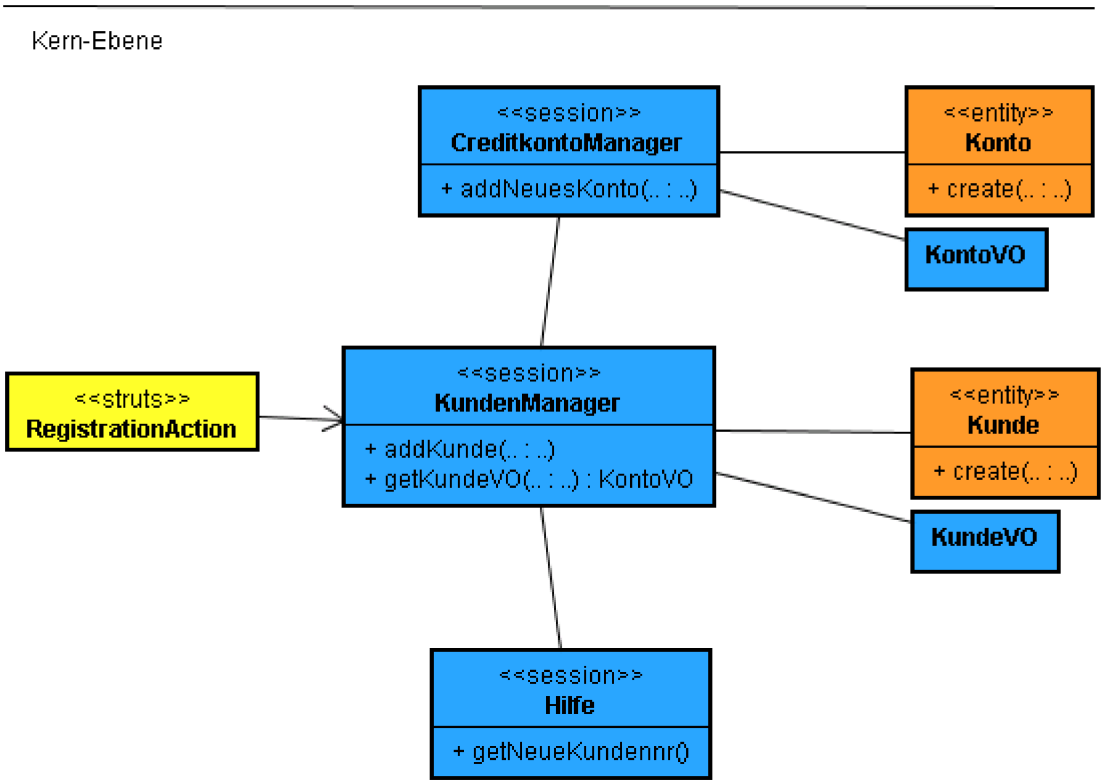

Abb. 33: Fassade und Verwendung von Value Objects

Die Fassaden-Klasse *KundenManager* wird aber nicht ausschließlich nur uni-direktional von anderen Klassen beliefert, sondern besitzt zusätzlich, wie fast alle Klassen des Geschäftsbereichs auf Kern- oder Service-Ebene, Schnittstellen-Charakter. Dies soll anhand der in Abbildung 33 aufgeführten Methode `getKundeVO(..)` der Klasse deutlich werden. Diese Methode liefert ein populiertes Value-Objekt eines bestimmen Kunden zurück. Die Methode wird sowohl von der Web-Infrastruktur, als auch ebenenübergreifend, von den speziellen Verwaltungsklassen der Services benutzt.

Auf Service-Ebene sei die Klasse *MobilAktionen* (Abb. 34) genannt, welche sich selbsterklärend im Paket *zentralfassade* befindet. Diese Klasse dient als zentraler und

einziger Anlaufpunkt für die Belange der mobilen Clients in allen Fragen der Serviceleistung, z.B. der des SMSmonkey-Services. Darunter fällt z.B. die Authentifizierung des mobilen Clients, bzw. des Kunden anhand ihrer Methode `authKunde(...)`. Die Zentralfassade unternimmt transparent für ihre aufrufende Klasse alle nötigen Schnitte, um den mobilen Client zu authentifizieren und retourniert nur eine Erfolgs- oder Fehlermeldung.

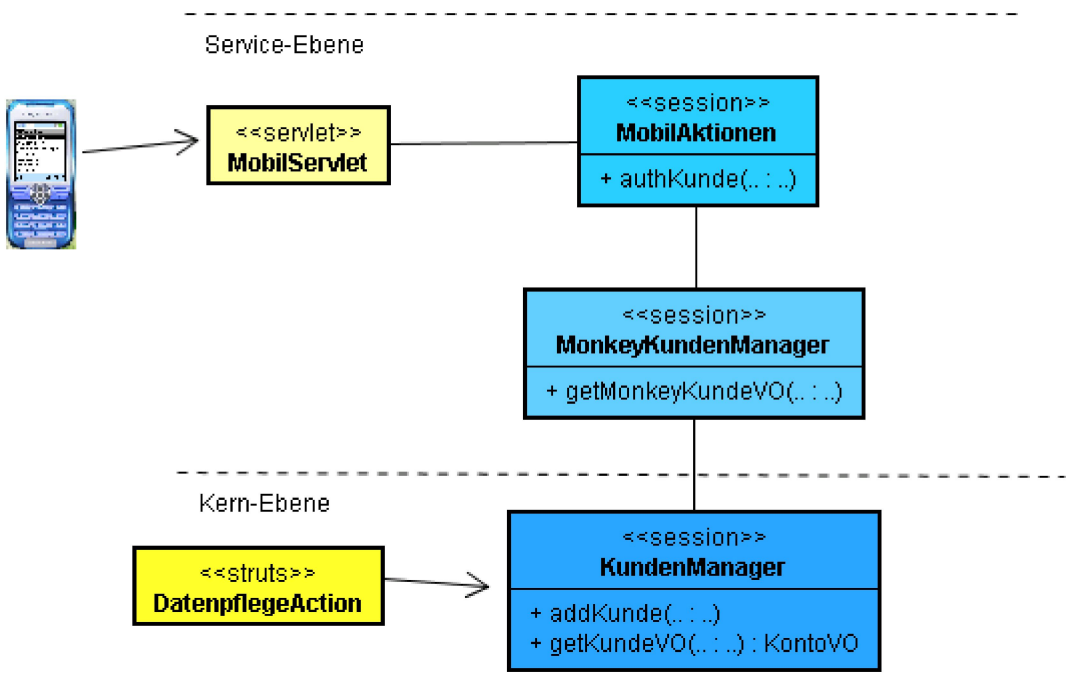

Abb. 34: Zentralfassade MobilAktionen

6.2 Datenobjekte

Um den Datenfluss innerhalb der nM-Plattform leichter zu handhaben, wird das in Kapitel 4.2 erläuterte Value Object Pattern angewendet. Es verkleinert die Signaturen von Methoden und verringert die Anzahl von Aufrufen zwischen den Komponenten innerhalb einer Ebene, sowie auch Ebenen-übergreifend. Zu jedem CMP Entity Bean existiert ein Value Object (VO), welches bei Bedarf von den Fassaden erzeugt und populiert wird. Als Beispiel sei hier die mit VOs in Zusammenhang stehende Methode `getKundeVO(..)` der Klasse *KundenManager* (Abb. 34) genannt. Die Methode kann dafür verwendet werden, einem Kunden zur Datenpflege seine Daten auf einer dem

entsprechenden Webseite (JSP) der Web-Infrastruktur anzuzeigen. Die KundenManager-Klasse bedient sich hierzu des Entity Beans *Kunde* der Datenhaltung und befüllt das korrespondierende Datenobjekt. Das Datenobjekt wird an die anfragende Struts-Action-Klasse zurück geliefert, wo die enthaltenen Daten weiterverarbeitet werden.

Aber nicht nur um lesenden Zugriff auf Kundendaten zu erhalten, sondern auch in anderer Richtung, von der Präsentation in Richtung Geschäftslogik, werden Datenobjekte verwendet: Die für den Versand von SMS zuständige Klasse besitzt u.a. eine Methode, die ein befülltes Datenobjekt vom Typ *SmsVO* erwartet und anhand deren SMS versendet werden.

Vor dem Hintergrund einer zukünftigen Weiterentwicklung der nM-Plattform sei angemerkt, dass dem Vorteil der Einsparung von Einzelaufrufen von Methoden und Vereinfachung von Methodensignaturen ein gewisses Fehler-Risiko beim Einsatz von Datenobjekten gegenüber steht. Es muss peinlich darauf geachtet werden, dass die Datenobjekte richtig und vollständig befüllt werden, um NullPointer-Fehler und Datentypen-Fehler zu vermeiden. An manchen Stellen ist es nicht möglich alle Attribute eines Datenobjekts zu befüllen, dann muss sich der Entwickler darüber bewusst sein und das Objekt dem entsprechend wieder korrekt auslesen. Es besteht die Gefahr von Flüchtigkeitsfehlern. Angemerkt sei ebenfalls, dass ein Datenobjekt immer nur eine Momentaufnahme des Zustandes des abgebildeten Objekts darstellt und nicht davon ausgegangen werden kann, dass das Original-Objekt des Entity Beans und das erzeugte Datenobjekt zu jedem Zeitpunkt in ihren Werten übereinstimmen.[26]

26 vgl. Adam Bien, Enterprise Java Frameworks, Addison-Wesley, 2001

7 Einsatz von JMX MBeans in der nM-Plattform

In der nM-Plattform sind bestimmte Session Beans als verwaltbare Ressourcen zu sehen. Diesen Ressourcen können zur Laufzeit Werte übermittelt werden und es können Werte von ihnen abgefragt werden. Dies geschieht durch aufzurufende Methoden, welchen Parameter übergeben werden können oder Werte retournieren und so die Initialisierungs- oder Konfigurationsaufgaben erfüllen.
In jeder logischen Ebene der nM-Plattform gibt es solch ein Session Bean, welches direkter Partner eines im JMX-Agenten bzw. im MBean-Server registrierten MBeans ist.

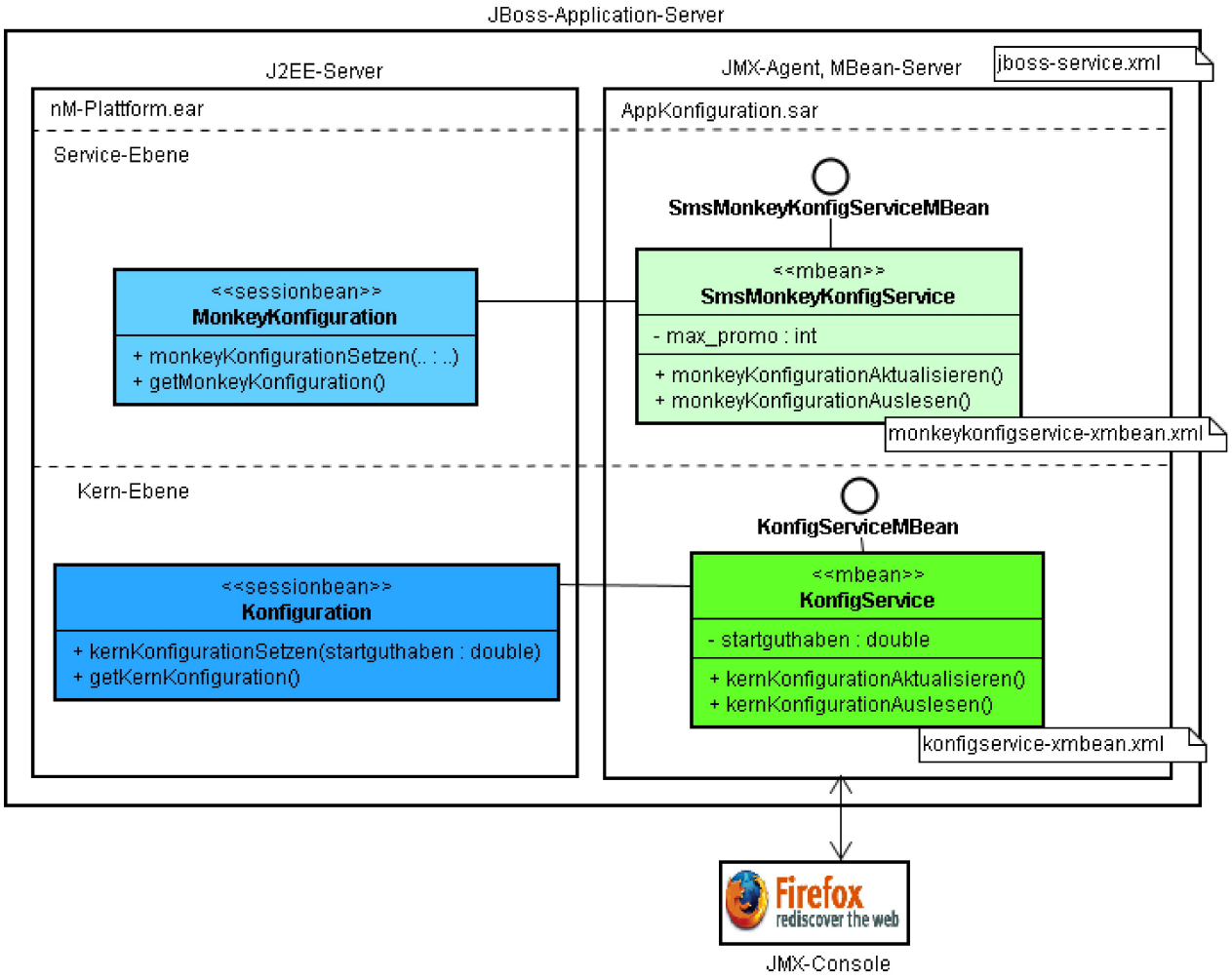

Abb. 35: Einsatz JMX-MBeans zur Konfiguration der nM-Plattform

Das Konzept der Konfiguration ist auf den verschiedenen Ebene der nM-Plattform

immer gleich, deshalb soll hier als Beispiel, stellvertretend für alle zur Laufzeit konfigurierbaren Werte, der Wert des Credit-Konto-Startguthabens eines neuen Kunden herangezogen werden. Für diesen Wert ist der Kern der nM-Plattform zuständig, da dort die allgemeine Kontoverwaltung erfüllt wird. Abbildung 35 soll die Trennung in Kern-und Servicekonfiguration darstellen und die Einordnung der Konfiguration erleichtern.

7.1 Das MBean

Das für die Konfiguration von Werten der Kern-Ebene zuständige ManagedBean ist das *KonfigService*-ManagedBean, bestehend aus der Klasse *KonfigService* und dem Interface *KonfigServiceMBean*, dem Management Interface. Dieses Interface erbt von der JMX-Implementation des JBoss Application Servers. Um ein MBean beim MBean-Server des JBoss registrieren zu können, bedarf es einer für alle MBeans zuständigen *jboss-service.xml* und einer MBean-spezifischen *konfigservice-xmbean.xml.* Die genannten XML-Dateien enthalten u.a. Angaben über Attribute und Methoden, welche über die Managementapplikation, in diesem Falle die *JMX Console Web Application* des JBoss, erreichbar und bearbeitbar sein sollen. Zur Namensgebung sei angemerkt, dass MBeans im JBoss als Service bezeichnet werden. Dem bin ich gefolgt.

```
<server>
...
  <mbean
code="de.netads.appkonfiguration.KonfigService" ... >

    <attribute name="Startguthaben">2.0
    </attribute>

      …
  </mbean>
...
</server>
```

Auszug 1: jboss-service.xml

In die *jboss-service.xml* (Auszug 1) werden alle zu konfigurierenden Attribute eines jeden MBeans eingetragen und können dort mit einem JMX-internen Default-Wert belegt werden.

Die MBean spezifische *konfigservice-xmbean.xml* (Auszug 2) beschreibt die zu konfigurierenden Werte feiner und es müssen dort alle Methoden eingetragen werden, die auf der MBean aufrufbar sein sollen.

```
...
      <attribute access="read-write" getMethod="getStartguthaben"
      setMethod="setStartguthaben">
            <description>Das Startguthaben des Credit-Kontos</description>
            <name>Startguthaben</name>
            <type>java.lang.Double</type>
            <descriptors>
            </descriptors>
      </attribute>
      ...
      <operation>
            <description>Übertragung der Kernparameter</description>
            <name>KernKonfigurationAktualisieren</name>
      </operation>

      <operation>
            <description>Auslesen der KernKonfigurationsWerte</description>
            <name>kernKonfigurationAuslesen</name>
            <return-type>java.lang.String</return-type>
      </operation>

...
```

Auszug 2: konfigservice-xmbean.xml

Die in der *konfigservice-xmbean.xml* angegebenen Methoden (Operations) `kernKonfigurationAktualisieren()` und `kernKonfigurationAuslesen()` der KonfigService-Klasse verantworten die Kommunikation mit der zu verwaltenden Ressource, dem Konfigurations-Session Bean auf Kern-Ebene. Das Session Bean *Konfiguration* besitzt zwei remote-Methoden, welche Partner der Kommunikationsmethoden der MBean sind und dessen Methoden vom Administrator über den Webbrowser anhand der JMX-Console aufgerufen werden können. Mit dem Aufruf der Methode `kernKonfigurationAuslesen()` ist es möglich in einem Aufwasch alle konfigurierbaren Werte aus dem Kern der nM-Plattform zu bekommen und im Browser zur Anzeige zu bringen, sie dort zu verändern und durch Aufruf der Methode `kernKonfigurationAktualisieren()`wieder an die J2EE-Applikation „zurück zu schicken“. Dort werden die Werte unter Verwendung der Fassaden

gesetzt, persistent gehalten und sind wiederum über die Fassaden dem Kern und den Services zugänglich. Der Austausch von Kern- und Service-Konfigurationsdaten in Richtung der MBeans geschieht unter Verwendung von Datenobjekten.

7.2 JMX-Console

Abb. 36: Startseite JMX-Console, Zugang zur Konfiguration der nM-Plattform

Die *JMX Console Web Application,* kurz JMX-Console (Abb. 36) ist die serverseitig laufende, JBoss-eigene Managementapplikation. Die MBeans und die nötigen beschreibenden XML-Dateien werden in einer *sar*-Datei gepackt auf dem JBoss deployed, welcher die MBeans dem JMX-Agenten bekannt macht und komfortabel über die JMX-Console zur Verfügung stellt. Anhand von Textfeldern kann auf die Attribute der MBeans zugegriffen werden und ein Administrator kann per JMX-Console Methoden wie z.B. `kernKonfigurationAuslesen()` (siehe Abb. 37) ausführen lassen. Diese Methode ist so gestaltet, dass die Attribute der MBean aktualisiert werden und die JMX-Console wiederum diese neuen Werte darstellt. In

anderer Richtung werden durch Veränderungen der Werte über die Textfelder und deren Übermittlung durch Aufruf z.B. der Methode `kernKonfigurationAktualisieren()` aktuelle Werte an die nM-Plattform-Applikation übertragen.

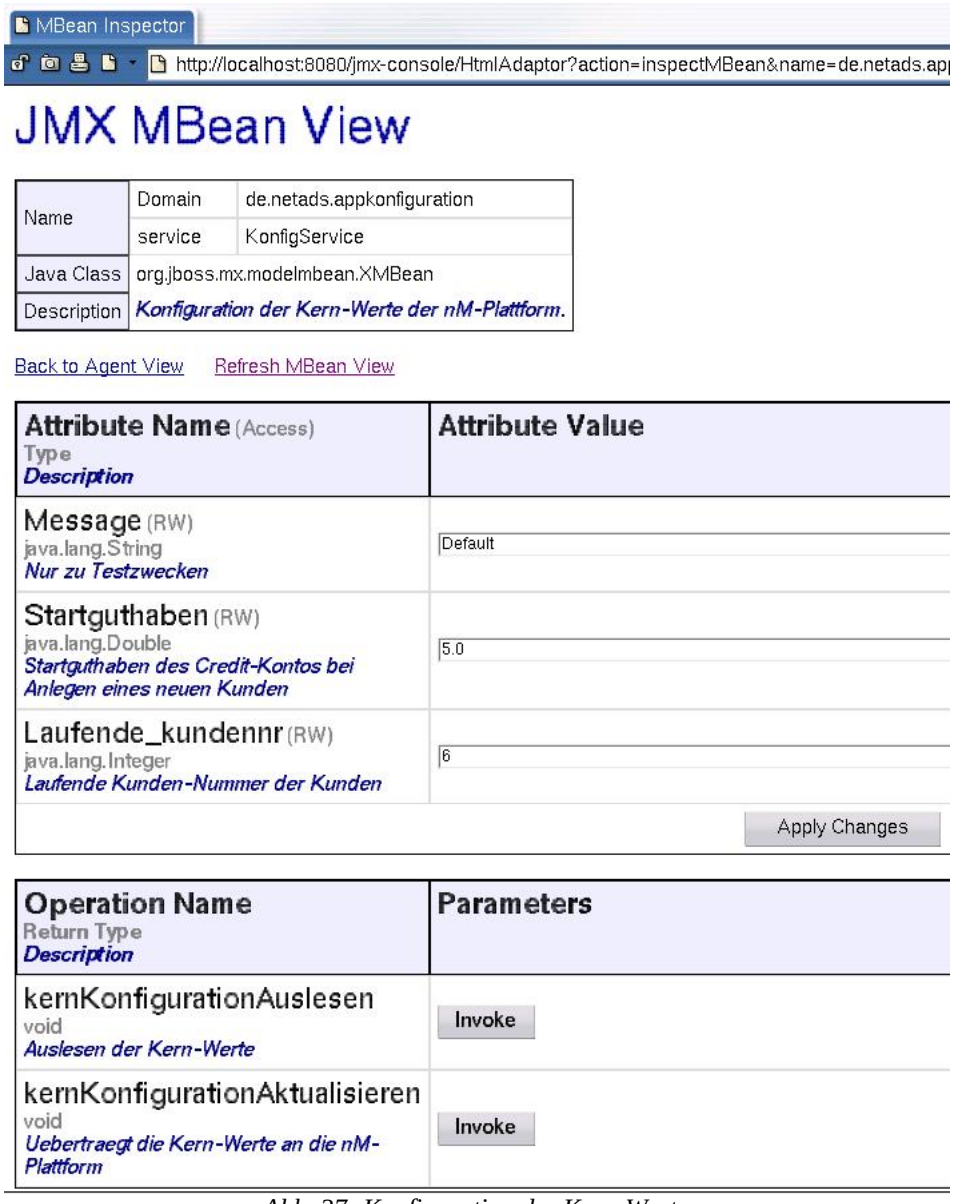

Abb. 37: Konfiguration der Kern-Werte

7.2.1 Sicherheit der JMX-Console

Um unerwünschte Zugriffe auf die *JMX Console Web Application* zu verhindern, ist diese durch eine Login-Passwort-Authentifizierung geschützt, welche in den Konfigurations-Dateien des JBoss-Servers gesetzt werden kann. Durch Austauschen der Management-Applikation der serverseitig laufenden *JMX Console Web Application* durch z.B. eine proprietäre, den RMI Konnektor nutzende standalone Management-Applikation, kann die Sicherheit der nM-Plattform noch einmal erhöht werden. Ist der JMX-Agent per Internet erreichbar, ist der Server immer einer potentiellen Gefahr ausgesetzt. Es könnten, z.B. eine Brute-Force-Attacke gegen die *Console Web Application* gefahren werden oder es wird versucht, je nach eingesetztem Konnektor, z.B. per RMI mit dem JMX-Agenten Kontakt aufzunehmen. Eine hohe Sicherheit kann mit der Verwendung von Firewall-Techniken erreicht werden, welche dafür sorgen, dass die JMX-Console nicht von Außerhalb des Fimennetzwerkes erreichbar ist.

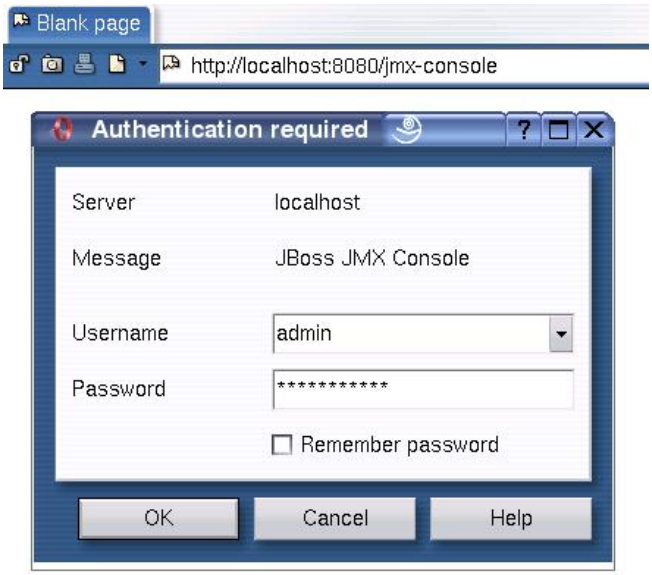

Abb. 38: Login zur JMX-Console

Ein Test mit einer Googel-Suche in der Form von `inurl:"jmx-console"` ergibt etwa 1500 Ergebnisse, viele davon eindeutig als JBoss JMX-Console identifizierbar.[27]

27 Stand 10.02.2005

Schon auf den ersten Seiten der Suchergebnisse gelingt es mehrmals ohne Authentifizierung auf eine JMX-Console zu gelangen. Dort können von allen und jedem die Bestandteile des JBoss umkonfiguriert und abgeschaltet werden. Zum Beispiel steht einem der Shutdown-Port und das dazugehörige Passwort des Embedded-Webcontainers zur „Konfiguration“ frei. Dies ist natürlich eine Katastrophe und man kann nur hoffen, dass betreffende Server nicht in einem aktiven Geschäftsprozess eingesetzt sind. Es ist also unbedingt erforderlich, den JBoss JMX-Agenten vor unerlaubten Zugriffen, je nach Budget mit Firewalls, Verschlüsselungstechnik oder zumindest mit der standard Login-Passwort-Authentifizierung zu schützen, da sonst erheblicher Schaden entstehen kann.

8 Erweiterung um einen neuen Service

Um die nM-Plattform um einen neuen Service zu erweitern wird der Referenz-Service, der SMSmonkey, herangezogen und nach dessen Vorbild die Applikation erweitert. Als Beispiel eines neuen Services soll hier der bei *netads* schon angedachte Bilderservice und dessen Einbindung in die nM-Plattform betrachtet werden. Der Bilderservice soll die ad-hoc Publikation von Bildern auf einer Webseite leisten, welche von einem mit Kamera ausgestatteten Mobiltelefon gemacht wurden.

Abbildung 39 soll den groben Aufbau eines Services der nM-Plattform andeuten und zeigen, dass ein neuer Service die selbe Grundstruktur wie der SMSmonkey-Service hat. Wie beim SMSmonkey-Service, bedarf es eines mobilen und eines webbasierten Anteils und es gibt ebenfalls mobile Clients und Webclients mit denen der Service kommuniziert. Die Daten von den Benutzerschnittstellen müssen kundenbezogen verarbeitet und dabei anfallende Kosten berechnet werden. Die Daten, im Fall des neuen Services Bilddaten, müssen gemäß der Leistungen des Services verarbeitet und gespeichert werden. Der Service muss, wie der SMSmonkey-Service auch, zur Laufzeit konfigurierbar sein.

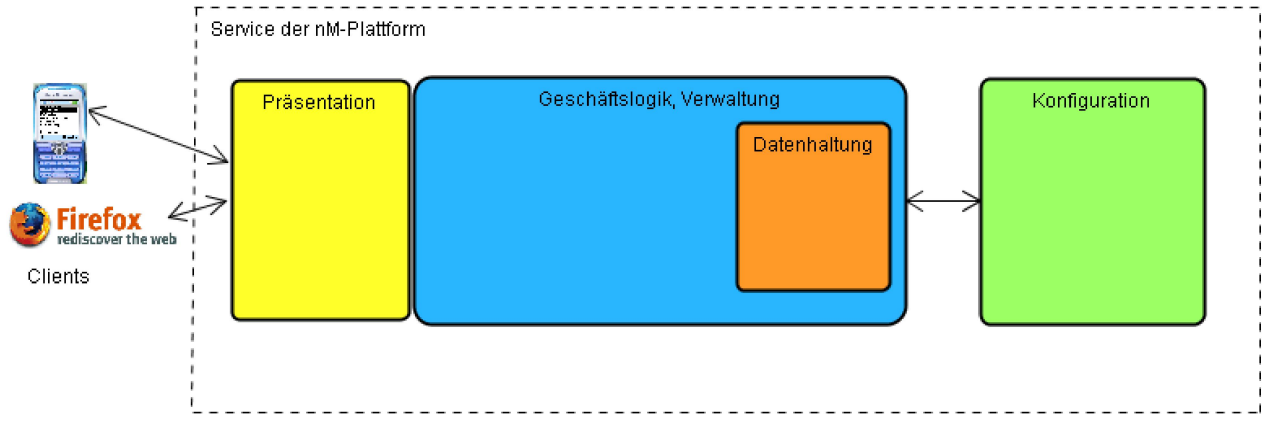

Abb. 39: Ein Service der nM-Plattform

Die Kern-Ebene der nM-Plattform nimmt dem neuen Service einen großen Teil der Kunden- und Kontodatenverwaltung ab und versorgt ihn mit Schnittstellen, z.B. für den Versand von SMS. Anhand der strukturellen Ähnlichkeit der Services kann die

Paketstruktur des SMSmonkey-Services übernommen werden und ein Großteil der Klassen, den neuen, speziellen Anforderungen des Services entsprechend, nachprogrammiert werden.

8.1 Präsentation

8.1.1 Web-Infrastruktur

Um sich beim neuen Bilderservice anzumelden, bzw. um ihn zu buchen, muss die webbasierte Infrastruktur der nM-Plattform erweitert werden. Ebenso wenn der Service webbasierte Leistungen, wie z.B. eine Bildergalerie bietet, bedarf es dem entsprechender JSPs, die zu erstellen sind. Um die webbasierte Präsentation umzusetzen wird das Paket `de.netads.kern.web` um ein Unterpaket erweitert. In ihm werden die zu den Webseiten (JSPs) korrespondierenden Struts-Action und Form-Klassen abgelegt, welche an die Geschäftsbereiche (EJBs) des neuen Services koppeln. Um den webbasierten Präsentationsteil zu vollenden, muss der neue Service in die Web-Infrastruktur der gesamten Applikation eingebunden werden. Dies geschieht anhand der deskriptiven Konfigurationsmöglichkeiten des Struts-Frameworks und dem Anpassen anderer JSPs. Vorstellbar wäre z.B. die Erweiterung einer Hauptseite um Hinweise und Verlinkung auf den neuen Service.

8.1.2 Mobile Kommunikation

Der neue Bilderservice hat, wie der SMSmonkey auch, einen mobiltelefonbasierten Anteil. Es werden mobile Clients existieren, mit denen der Service kommunizieren muss. Gemäß der Referenz des SMSmonkey-Services bedarf es einem Paket `de.netads.neuerservice.mobilfrontend`, in dem sich die Klassen der direkten Kommunikation mit dem mobilen Clients befinden. Soll der Datenfluss zu und von den Webclients und/oder den mobilen Clients überprüft, gefiltert oder verändert werden, kommen Klassen zum Einsatz, welche im Paket `de.netads.kern.filter` anzulegen sind. Sind in diesem Paket schon geeignete

Filterklassen vorhanden, können diese benutzt werden.

8.2 Geschäftsbereich

Der Geschäftsbereich teilt sich in mehrere Pakete auf, die in Anlehnung an den SMSmonkey-Service angelegt werden müssen:

```
de.netads.neuerservice.verwaltung
de.netads.neuerservice.zentralfassade
de.netads.neuerservice.service
de.netads.neuerservice.konfiguration
de.netads.neuerservice.hilfe
```

- de.netads.neuerservice.zentralfassade

Wie der SMSmonkey-Service auch, sollte jeder neue Service an dieser Stelle eines oder mehrere Session Beans haben, welche als Fassaden zwischen dem Präsentationsteil und dem Geschäftsbereich stehen. Hier laufen alle Belange der mobilen Clients auf und können vorverarbeitet werden.
Die Fassade bündelt und kanalisiert die Kommunikation zwischen den mobilen Clients und der Geschäftslogik, enthält aber auch schon selbst einen kleinen Teil der Logik, was die tiefer im Geschäftsbereich liegenden, Service-spezifischen Klassen genereller ausfallen läßt. Die Struts-Action-Klassen und die Zentralfassade dienen als Adapter der puren Geschäftslogik zu den Clients.

- de.netads.neuerservice.service

In den Klassen dieses Paketes wird die nackte Geschäftslogik des neuen Services umgesetzt und bedient Zentralfassade und Service-spezifische Web-Infrastruktur.
Beim Beispiel des Bilderservices wird hier die Verarbeitung von Bilddaten stattfinden.

- de.netads.neuerservice.verwaltung

An dieser Stellen werden Klassen der Kunden- und Kontodatenverarbeitung des neuen Services angelegt. Wie beim SMSmonkey-Service werden hier generelle Verwaltungsdaten aus der Kern-Ebene der nM-Plattform zu Service-spezifischen

Daten umgesetzt und ergänzt. Die Klassen sind Zwischenschicht zur allgemeinen Verwaltung im Kernbereich.

- de.netads.neuerservice.konfiguration

Bei der Planung des neuen Services müssen die Werte herausgestellt werden, welche zur Laufzeit der nM-Plattform-Applikation veränderbar sein sollen. Nach Vorbild des SMSmonkey-Services leisten die in diesem Paket angelegten Klassen die Kommunikation mit dem JMX-Bereich und die Verteilung der Daten an die Entity Beans der Datenhaltung. Ein Klasse mit Service-spezifischen Konstanten nach Vorbild von Kern und SMSmonkey-Service befindet sich ebenfalls hier.

- de.netads.neuerservice.hilfe

Bei Bedarf und nach Vorbild des SMSmonkey-Services stellt das Paket `hilfe` eines neuen Services einen Platz für unterstützende Klassen dar, deren Leistungen sonst keinem Bereich bzw. Paket zugeordnet werden können, ohne deren strukturelle Harmonie zu stören.

8.3 Datenhaltung

Die Datenhaltungsaufgaben werden, wie im Kern oder dem SMSmonkey-Service, von CMP Entity Beans geleistet, welche sich im jeweiligen Pakte `persistenz` befinden. Dem entsprechend wird für einen neuen Service ein Paket `de.netads.neuerservice.persistenz` angelegt. Die zu den Entity Beans korrespondierenden Value-Object-Klassen und andere Datenobjekte des neuen Services haben ihre Heimat in `de.netads.neuerservice.datenobjekte.`

8.4 Erweiterung der Konfiguration

In Kapitel 7 wird erläutert, wie JMX in der Konfiguration der nM-Plattform eingesetzt wird. Nach diesem Beispiel und in gleicher Weise wie die Konfiguration des SMSmonkey-Services, muss der JMX-Teil der nM-Plattform-Applikation erweitert

werden.

Es muss für den neuen Service ein MBean entwickelt werden, welches mit den verantwortlichen Klassen im Paket `de.netads.neuerservice.konfiguration` des neuen Service kommuniziert. Hierzu muss die *jboss-service.xml* des JMX-Teils erweitert und ein neue *xmbean.xml*-Datei angelegt werden. In diesen Dateien werden Kommunikations-Methoden und die variablen Werte des neuen Services beschrieben.

8.5 Übersicht

Der Gesamtüberblick über die Pakete des neuen Services und des Referenzservices stellt sich wie in Abbildung 40 dar.

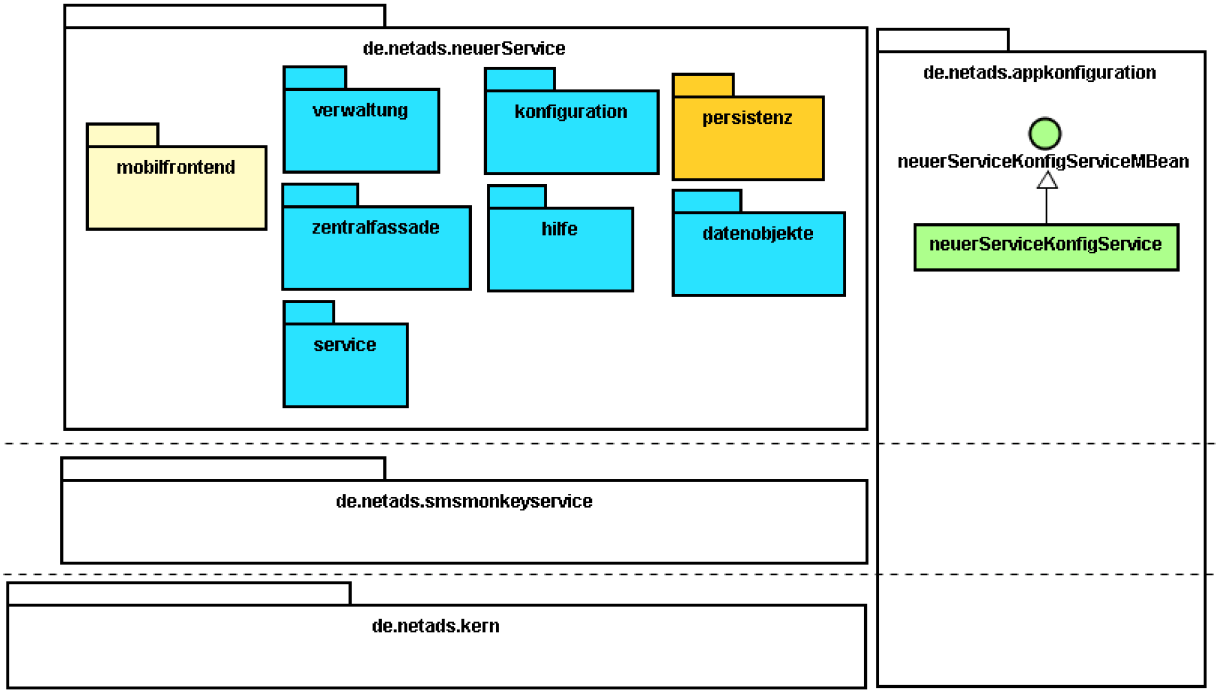

Abb. 40: Ein neuer Service in der nM-Plattform

Die Andeutung des Kern-Hauptpaketes und der Pakete des SMSmonkey-Services sollen noch einmal die Ebenenstruktur der nM-Plattform verdeutlichen. Abbildung 40 zeigt auch die Erweiterung des JMX-Konfigurations-Bereichs um jeweils ein Interface und eine Klasse, das MBean des neuen Services.

9 Sicherheit

Im folgenden Kapitel wird beschrieben, welchen Gefährdungen die nM-Plattform-Applikation ausgesetzt ist und was unternommen wird, um diese auszuschalten oder so gering wie möglich zu halten.

9.1 Einsatz neuster Softwareprodukte

Die bei der nM-Plattform eingesetzten und potentiell gefährdeten Softwareprodukte (z.B. Web- und Application Server, DBMS) sind auf einem hohen Versionsstand und waren zu Beginn der Realisierung nicht älter als 6 Monate. Damit sollte gewährleistet sein, dass die meisten der bis zum Erscheinen der eingesetzten Version bekannten Sicherheitslücken von den Herstellern ausgemerzt wurden. Ein Programmierer, welcher die nM-Plattform weiterentwickelt, muss sich regelmäßig über Updates und Pachtes informieren, um die Applikation möglichst sicher zu halten.

9.2 Umgang mit Fehlermeldungen

Bei der Implementierung der Applikation wurde darauf geachtet, dass eine Trennung zwischen applikationsinternen und für den Kunden bestimmte Fehlermeldungen stattfindet.
Es wäre fatal, wenn ein Angreifer bei der Applikation Fehler provozieren könnte, die in für ihn sichtbare Fehlermeldungen enden, anhand deren auf die Eigenschaften und den Aufbau des Systems geschlossen werden kann. Es wird also darauf geachtet, dass Fehler, die der Benachrichtigung des Kunden bedürfen, in für Kunden geeignete Informationen umgesetzt werden und applikationsinterne Fehler, die den Kunden nicht zu interessieren haben, nicht bis zum Kunden vordringen. Es könnte z.B. sein, dass im Zuge von Erweiterungsarbeiten an der nM-Plattform von einem Programmierer übersehen wird, eine bestimmte Exception zu behandeln. Wird diese dann irgendwann geworfen, wandert sie in entgegengesetzter Richtung entlang der

Kette der aufrufenden Methoden, in der Hoffnung irgendwo behandelt zu werden. Geschieht dies nicht und die Exception läuft bis in den Web Tier, kann es sein, dass sie dem Benutzer angezeigt wird, was zu vermeiden ist. Um im webbasierten Teil der nM-Plattform in letzter Instanz einen Schutz vor solchen denkbaren, da menschlichen Fehlern zu haben, wird der *web.xml* des Webservers folgender Eintrag hinzugefügt:

```
<error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/error.jsp</location>
</error-page>
```

Dieser Eintrag sorgt dafür, falls eine Exception den Webserver erreicht hat, an der Stelle des Fehler-Textes der Exception eine *error.jsp* angezeigt wird, welche undetaillierte Informationen für den Kunden enthält.

Auf Seiten der für die mobilen Clients zuständigen Servlets hätte eine unbehandelte und nicht in regelkonforme Meldungen umgesetzte Exception weitere Auswirkungen, da das Zurückliefern von definierten Fehlercodes an die mobilen Clients scheitern und somit der Programmablauf der mobilen Clients gestört werden könnte. Deshalb ist in der Kommunikation mit den mobilen Clients besonders darauf zu achten, dass Fehler korrekt behandelt werden und dem entsprechende Meldungen geliefert werden.

9.3 Sicherheit durch Validierung und Filter

Die von Benutzern an die Applikation gerichteten Geschäftsinformationen müssen darauf hin überprüft werden, ob sie gewissen Regeln entsprechen. Geschäftsinformation werden der nM-Plattform zum Einen über die mobilen Clients und zum Anderen über Webbrowser zugeführt.

Als Beispiel sei hier der Benutzername eines Kunden angeführt und die darauf anzuwendende Regel, dass dieser nicht kürzer als vier Zeichen sein darf. Möchte sich ein Kunden an der nM-Plattform registrieren, um einen Service nutzen zu können, muss er einen Benutzernamen wählen und ihn auf einer Webseite in das entsprechende Formularfeld eingeben. Nach Absenden des Formulars werden die Angaben anhand des Struts Validation Frameworks auf dessen Längen hin überprüft. Entspricht dieser nicht der Regel, wird die Eingabe nicht angenommen und es

erscheint eine Fehlermeldung. Damit ist sichergestellt, dass alle neuen Kunden einen regelkonformen Benutzernamen haben.
Der Benutzername taucht noch einmal bei der Freischaltung der mobilen Clients auf, im speziellen Fall zum Freischalten des MIDlets des SMSmonkey-Services. Das MIDlet kommuniziert mit der Klasse *MobilServlet*, ein Servlet welches für den Datenaustausch mit dem MIDlet verantwortlich ist.

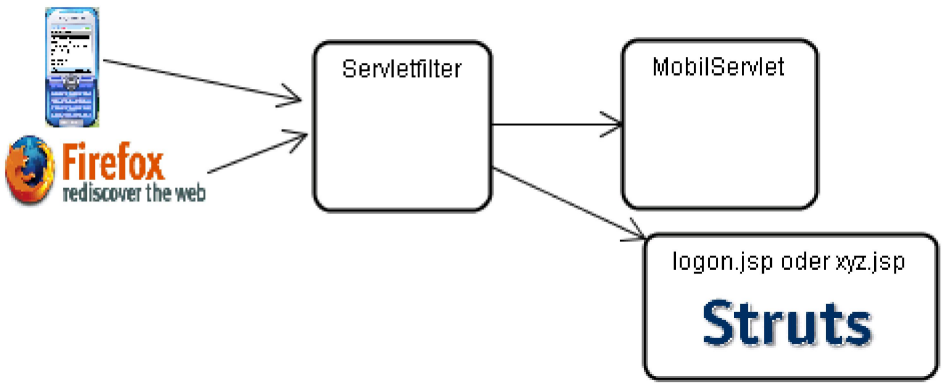

Abb. 41: Filterung von Geschäftsdaten

Die Parameter des HTTP-Requests werden davor in einem Servletfilter überprüft (Abb. 41). Darunter fällt auch der beim Freischalten anzugebende Benutzername. Ergibt die Überprüfung z.B. der Länge der Zeichenkette des Benutzernames eine andere, als die festgelegte, könnte es sich um den Versuch handeln, den Datenverkehr zwischen dem MIDlet und der nM-Plattform nachzustellen. Die weitere Verarbeitung des Requests wird abgebrochen und von dem Servletfilter eine Fehlermeldung an den Client zurückgeliefert.
Wie ersichtlich ist, können anhand der Struts Validation und den in Servletfiltern implementierten Prüfungen alle, von Benutzern per Web oder von mobilen Clients übertragenen Geschäftsinformationen, überprüft werden. Der Grad der Sicherheit vor eventuell schädlichen Zeichenketten richtet sich nach der Feingranularität der auf die Zeichenketten angewendeten Regeln.

9.4 Zentraler Zugang und Authentifizierung

Um das Webangebot der Services auf der nM-Plattform nutzen zu können, muss sich ein Benutzer zuerst mit Benutzername und Passwort anmelden bevor er ein Angebot nutzen kann. Die nM-Plattform ist so gebaut, dass alle Anfragen an den Webserver der nM-Plattform durch Servletfilter laufen (Abb. 41). Dies ist in der *web.xml* des Webservers definiert. Der Servletfilter dient als der zentrale und einzige Eingang zum Webserver der nM-Plattform.

In einem ersten Schritt wird dort geprüft, ob die HTTP-Anfrage an den Server von einem mobilen Client oder von einem Webbrowser stammt. Dies geschieht durch die Analyse der HTTP-Request-Parameter. Ergibt die Analyse, dass die Anfrage von einem mobilen Client stammt, werden Zeichenenkettenprüfungen durchgeführt und der Request bei positivem Ergebnis dem weiter verantwortlichen *MobilServlet* übergeben. Wurde festgestellt, dass die Anfrage von einem Webbrowser stammt, wird überprüft, ob der Kunde schon eingeloggt ist oder nicht. Ist er es nicht, wird er an die Seite *logon.jsp* weitergeleitet, auf der es sich einloggen kann.

Die Struts-Action-Klasse, welche die Benutzer- und Passwortdaten des Kunden aus den Textfeldern der *logon.jsp* überreicht bekommt (*LogonAction*) bedient sich der Verwaltungsfassade des Kerns, um den Kunden zu authentifizieren. Ist die Authentifizierung erfolgreich, speichert die Struts-Action das Datenobjekt des Kunden als Attribut in die Session des Requests. Die Prüfung im Servletfilter, ob ein Kunde sich erfolgreich eingeloggt hat oder nicht, findet dies heraus, in dem geschaut wird, ob sich in der aktuellen Session ein Attribut mit der Bezeichnung befindet, welches dem von der LogonAction-Klasse hinzugefügten entspricht. Weiter wird geprüft ob der Wert des Attributes ein gültiges Objekt ist. Sind diese beiden Bedingungen erfüllt, wird der Kunde auf die gewünschte Seite des Web-Bereichs der nM-Plattform geleitet. Ist dies nicht der Fall, wird ihm die *logon.jsp* präsentiert, wo dem Besucher auch die Möglichkeit angeboten werden kann, sich als neuer Kunde zu registrieren.

Der Filter- und Authentifizierungsmechanismus per Servletfilter hat einige Vorteile:

- Völlige Entkopplung von Security und Geschäftsaufgaben
- Zusammenfassung und Auslagerung der Securityaufgaben in die Servletfilter

- Die Servletfilter-Lösung ist unabhängig vom Struts-Framework. Weder gibt es eine enge Verzahnung mit Struts, noch ist dessen eigenhändige Erweiterung nötig, um Filter zu realisieren.

9.5 Speicherung von Passwörtern

Passwörter werden von der nM-Plattform-Applikation nicht im Klartext gespeichert. Die Passwörter werden verschlüsselt abgelegt. Bei einem Login-Vorgang, dem Freischalten von mobilen Clients oder anderen Bedarfsfällen wird das angegebene Passwort ebenfalls verschlüsselt und mit der gespeicherten Zeichenkette verglichen.

9.6 SQL-Injections

SQL-Injections sind eine Art von Angriff gegen Datenbank-basierte Applikationen. Erfolgreiche Angriffe können es einem Angreifer ermöglichen, sich z.B. unerlaubt in eine Web-Applikation mit Zugriffsschutz einzuloggen, Informationen aus einer Datenbank zu ermitteln oder Daten in einer Datenbank zu manipulieren.

Bei Mehrschicht-Applikationen, wie der nM-Plattform auch, werden Daten in Form von Zeichenketten von den Clients an die verarbeitende Geschäftsebene und danach an die Datenebene zur Speicherung oder zur Bildung von Abfragen durchgereicht. Letztendlich werden auf der Datenebene die Geschäftsinformationen aus der Präsentationsschicht in SQL-Statements eingebettet und an die Datenbank geleitet. Ein Angreifer kann auf diesem Wege anstatt der erwarteten Zeichenketten an Geschäftsinformationen auch SQL-Code an den Server senden, welche dieser dann wie gewöhnliche Geschäftsinformationen in SQL-Statements einbettet. Der Angreifer kann dadurch die ursprüngliche Semantik der SQL-Statements verändern und so erheblichen Schaden anrichten.
Schwerwiegende Ziele, die Angreifer anhand von SQL-Injection-Angriffe bei ungeschützten Applikationen verfolgen und erreichen können, sind die Umgehung der Authentifizierung der Applikation (z.B. das Einloggen als ein beliebiger Benutzer) und die Ausführung von SQL-Statements im DBMS. Dies ist besonders gefährlich, da

bei einem erfolgreichen Angriff auch löschende und Daten-verändernde SQL-Anweisungen ausgeführt werden könnten.[28]

Im allgemeinen, wie auch im speziellen Falle der nM-Plattform, ist der Web-Applikations-Teil besonders gefährdet, da dieser ständig von beliebig vielen Personen per Internet erreichbar ist. Die mobilen Clients der Plattform werden zwar von einer geschlossenen Benutzergruppe verwendet, da aber die Kommunikation hier ebenfalls über HTTP geschieht und aus Klartext-Zeichenketten besteht, könnten die Geschäftsinformationen eines mobilen Clients nachgestellt werden und ebenfalls SQL-Injection-Angriffe enthalten.
Als einfaches Beispiel eines SQL-Injection-Angriffs soll hier die Umgehung der Authentifizierung und Einloggen als ein User mit dem Username `arthur` ohne Kenntnis eines Passwortes über das Login/Passwort-Fenster einer Webseite angeführt werden. Das Beispiel bezieht sich auf keine spezielle Applikation und soll nur verdeutlichen, wie ein Angriff aussehen könnte und was für Maßnahmen dagegen ergriffen werden können.
Um sich in die Applikation einzuloggen wird die Angabe `Username` und `Passwort` benötigt, die z.B. wie folgt aussehen könnten:

```
Username: arthur
Passwort: dent_5?343-x
```

Die fiktive Beispielapplikation ist nun so programmiert, dass die angegebenen Geschäftsinformationen direkt in eine SQL-Abfrage einfließen. Das daraus resultierende SQL-Statement würde so aussehen:

```
SELECT * FROM users
WHERE username = 'arthur'
AND password = 'dent_5?343-x'
```

Damit ein Kunden sein Passwort maximal sicher gestalten kann ist die Beschränkung dessen Inhalts auf Ziffern und Buchstaben zu wenig, also werden Sonderzeichen

28 vgl. http://www.infsec.ethz.ch/people/psevinc/sqliadReport.pdf, 31.01.05

zugelassen, und diesen Zustand nutzt ein Angreifer aus.
Im obigen Beispiel wird die Zeichenkette der Geschäftsinformation `arthur` und `dent_5?343-x` im SQL-Statement mit einfachen Hochkommata (') begrenzt, was dem ANSI-SQL-Standard entspricht.[29] Ein Angreifer könnte aber nun für die Felder `Username` und `Password` die folgenden Werte eingeben:

```
Username: (leer)
Password:' OR username = 'arthur
```

Die Zeichenkette `username` als Bezeichnung der Tabellenspalte der Benutzernamen in der Datenbank kann geraten sein oder durch gezielte Aktionen in Erfahrung gebracht worden sein. Durch die automatische Begrenzung der hier schädlichen Geschäftsinformationen mit einfachen Hochkommata zu standard-konformem SQL resultiert das folgende, legale SQL-Statement:

```
SELECT * FROM users
WHERE username = ''
AND password = '' OR username = 'arthur'
```

Durch das angegebene OR ist hier nun der Ausdruck *username = 'arthur'* für die Erfüllung der ganzen Bedingungen maßgebend. Dadurch wird erreicht, dass die Bedingungen für den Datensatz mit dem Wert `arthur` im Feld `username` erfüllt sind.
Der Angreifer wäre in der fiktiven Beispielapplikation nun als Benutzer `arthur` eingeloggt, falls ein Benutzer mit diesem Benutzernamen existiert.
Wie das Beispiel zeigt, ist das Zulassen von Sonderzeichen in Passwörtern, ins besondere von einfachen Hochkommata (´), problematisch. Solche Zeichen müssen unschädlich gemacht werden. Dies ist nicht ganz einfach, wenn sie in der Eingabe trotzdem zugelassen werden sollen. Um eine Applikation nun sicher gegen SQL-Injection-Angriffe zu machen, gleichzeitig die Verwendung von kritischen Sonderzeichen zuzulassen und gleichzeitig nicht von einem Datenbankprodukt abhängig zu sein, müßte ohne Hilfsmittel ein enormer programmatischer Aufwand

29 vgl. http://www.infsec.ethz.ch/people/psevinc/sqliadReport.pdf, 31.01.05

betrieben werden.[30]

9.6.1 Prepared Statements

Genannte Problematik der SQL-Injections kann mit dem Einsatz von Prepared Statements in den Griff bekommen werden. Prepared Statements sind parametrisierte SQL-Anweisungen.
Ein typisches Prepared Statement würde etwa so aussehen:

```
SELECT * FROM stadt WHERE plz = ?
```

Prepared Statements werden zunächst deklariert und zum Vorkompilieren an das Datenbanksystem übergeben. Anschließend können sie dann ausgeführt werden, indem die formalen Parameter (?) durch aktuelle Werte ersetzt werden und die so parametrisierte Anweisung an die Datenbank übergeben wird .
Es entsteht also eine Auftrennung der Abfrage in einen puren SQL-Anweisungsteil und der Zeichenketten mit den Geschäftsinformationen.
Die bösartig gestaltet Zeichenkette kann unter diesen Voraussetzungen das SQL-Statement nicht mehr verändern und es kann somit kein Schaden angerichtet werden, da die Geschäftsinformation, ob bösartig oder nicht, verarbeitet wird wie sie ist.
Die Prepared Statement-Technik ist in den Datenbanktreibern der Hersteller implementiert oder in der Datenbank selbst.[31]
Im Falle der nM-Plattform leistet der Java-MySQL-Datenbanktreiber *Connector-J* in der Version 3.1 die Umsetzung der Prepared Statement-Technik.[32] [33]

9.7 Sicherheit durch JMX

Der in Kapitel 3.5 und 7 erläuterte Einsatz der Java Management Extention hat den weiteren Vorteil einer erhöhten Sicherheit der nM-Plattform-Applikation. In vielen Web-Applikationen ist es üblich, dass sich ein Administrator über den normalen Login-Passwort-Mechanismus, über den sich auch Kunden anmelden, Zugang zum

30 vgl. http://www.infsec.ethz.ch/people/psevinc/sqliadReport.pdf, 31.01.05
31 vgl. http://dev.mysql.com/tech-resources/articles/4.1/prepared-statements.html, 31.01.05
32 vgl. http://www.galileocomputing.de/openbook/javainsel4/javainsel_20_011.htm, 31.01.05
33 vgl. http://dev.mysql.com/doc/connector/j/en/index.html, 31.01.05

System verschafft, um administrative Arbeiten auszuführen. Der „Kunde“ Administrator ist gegenüber normalen Kunden nur mit mehr Rechten ausgestattet und hat dadurch Zugriff auf Administrations-Seiten der Applikation. Diese Vorgehensweise birgt ein gewisses Risiko, wie unter Betrachtung des Kapitels der SQL-Injections gefolgert werden kann. Die Angriffsfläche um Schaden anzurichten ist über einen zentralen, für Jedermann sichtbaren Adminitrationszugang sehr groß. Beim Einsatz von JMX ist die Administration vom Business der Applikation völlig abgetrennt und eigenständig und kann über das Kunden-Frontend nicht erreicht werden. Den zuverlässigen Schutz des Zugriffs auf den JMX-Agenten vorausgesetzt, hat ein potentieller Angreifer sehr hohe Hürden zu überwinden, um Zugriff auf informationsträchtige Daten oder zerstörerisch einsetzbare Administrations-Funktionen zu erhalten. Sollte ein Angreifer dennoch Zugang über das Kunden-Frontend erhalten, hat dieser lediglich die gleichen, fest definierten Möglichkeiten wie ein legaler Kunde und könnte in diesem Rahmen Schaden verursachen, welcher aber im Vergleich der Möglichkeit zur Zerstörung der Applikation oder der unbemerkten Veränderung von sensiblen Konfigurationswerten nur gering wiegt.

10 Eingesetzte Software

Eine der Vorgaben bei der Entwicklung der nM-Plattform und der Portierung des SMSmonkey-Service war die möglichst kostengünstige Entwicklung. Es sollte möglichst nur freie, kostenlose Software zum Einsatz kommen, was auch durchweg gelang, da alle Anforderungen, welche die Entwicklungsaufgabe mit sich brachte, von der freien Software erfüllt wurden.

Zum Einsatz kamen folgende Komponenten:

Betriebssystem:	Suse Linux 9.0
Datenbank:	MySQL 4.0
Application Server:	JBoss 3.2.3
Webcontainer:	Tomcat 4.0 (in JBoss eingebettet)
Entwicklungsumgebung:	Eclipse 3M6
	Plugins: Lomboz

10.1 Betriebssystem Linux

Zum Einsatz kam eine Suse Linux 9.0 professional Distribution. Die Hauptargumente für deren Gebrauch lagen darin, dass Linux kostenlos ist und der Gebrauch von MySQL als freies Datenbankmanagementsystem favorisiert war. Dies setzte nämlich den Einsatz von Linux voraus, da MySQL nur unter Linux, nicht aber unter Microsoft-Betriebssystemen kostenlos ist.[34] Die Wahl von Linux als Betriebssystem ermöglicht die Entwicklung und das lokale Testen des Systems unter der selben oder zumindest einer versionsnahen Linux-Distribution, wie auf einem später gehosteten Server und ermöglicht eine weniger reibungsvolle Verlegung des Systems auf den späteren Host. Auch zu Zwecken der Weiterentwicklung kann so ein fast identisches System kostenfrei lokal betrieben werden. Allgemein bekannte Argumente für den Einsatz von Linux sind u.a. dessen Stabilität, Konfigurierbarkeit und höhere Sicherheit im Gegensatz zu Microsoft-Produkten.

34 vgl. http://www.mysql.com, 24.01.05

10.2 Datenbanksystem MySQL

MySQL ist unter Linux auch für kommerzielle Anwendungen ein freies Datenbankmanagementsystem. MySQL ist bekannt und weit verbreitet, und es gibt sehr viele Webseiten und Foren, die sich mit MySQL beschäftigen, was einem die Suche nach Informationen und Hilfe erleichtert. Weiter existieren ein Reihen von ebenfalls freien Tools, die einem die Einrichtung und Administration der Datenbank erheblich erleichtern. Mit MySQL lassen sich kostengünstige aber dennoch leistungsstarke Internet-basierte Datenbankapplikationen aufbauen. MySQL ist äußerst stabil und auch bei großen Datenmengen extrem schnell und effizient. MySQL verwaltet Indizes, wie andere Datenbanken auch, in Baumstrukturen. Auf diese Datenstrukturen kann mit logarithmischer Komplexität zugegriffen werden, d.h. es sind nur sehr wenige Zugriffe notwendig, um eine beliebigen Zieldatensatz über den Index zu finden.[35] Auf Geschwindigkeitsoptimierung haben die Entwickler auch sonst großen Wert gelegt, was sich an den letzten Benchmark-Veröffentlichungen auf der MySQL-Webseite widerspiegelt.[36] In besagtem Benchmark-Test wird auch auf die sehr hohe Stabilität von MySQL hingewiesen.
Als Verwaltungstool kam das freie *MySQL Administrator* zum Einsatz, welches einem Administrator anhand einer grafischen Benutzeroberfläche u.a. erlaubt, Datenbanken anzulegen und zu manipulieren. *MySQL Administrator* steht in der Version 1.0 zum Download bereit.[37] Ein wichtiger Aspekt des Tools ist es, auf komfortable und vielseitige Weise Datensicherung durchzuführen und gesicherte Daten im Notfall einfach wieder einzuspielen.

10.3 Application Server JBoss

Bei der Entwicklung der nM-Plattform kam ein JBoss der Version 3.2.3 mit eingebettetem Webserver Tomcat 4.0 zum Einsatz.
JBoss ist ein vollständig J2EE 1.3 kompatibler Open-Source Server, der unter der LGPL (Lesser General Public Licence) frei für den kommerziellen und privaten

35 vgl. Dicken,Hipper,Müßig-Trapp, Datenbanken unter Linux, Mitp, 2000
36 vgl. http://dev.mysql.com/tech-resources/benchmarks, 15.11.04
37 siehe http://www.mysql.com/products/administrator, 25.01.05

Gebrauch angeboten wird. Im Moment liegt JBoss in der Version 4.01 vor.[38] Bestandteil der JBoss Distribution sind unter anderem der EJB-Container, der Ablaufumgebung für Enterprise Beans, Jetty oder Tomcat als Webcontainer, Hypersonic SQL als eingebaute Datenbank und JBoss MQ als nachrichtenorientierte Middleware. Es ist somit möglich JBoss ohne weitere Softwarekomponenten zu betreiben. Im Kern der JBoss-Implementierung befindet sich ein JMX (Java Management Extension) Server, der nach dem Programmstart alle konfigurierten Bestandteile lädt, initialisiert, startet und diese verwaltet. Das Management der Bestandteile erfolgt über JMX. Hierzu gibt es zwei Adapter, die sich per RMI oder HTTP ansprechen lassen. Das standard-JMX-Webinterface ist unter http://Server:8080/jmx-console/ zu erreichen. Hier lassen sich alle Betriebszustände und die Konfigurationen von Server-Bestandteilen verwalten. Im deploy-Verzeichnis des JBoss können eigene Programme abgelegt werden. Sie werden dort automatisch, auch wenn JBoss gestartet ist (Hot Deploy), erkannt und geladen. Analog dazu werden die Anwendungen beendet, wenn das Archiv wieder gelöscht wird. Im Deploy-Verzeichnis eines JBoss Application Servers befinden sich auch alle XML-Dateien, welche Dienste beschreiben (Deployment Descriptor), und SAR-Archive, die eine JBoss Besonderheit sind und JMX-Dienste enthalten.[39] [40]

10.3.1 Webserver Tomcat

Der JBoss Application Server unterstützt die Einbindung von Webcontainern per JMX. In der eingesetzten Version 3.2.3 des JBoss kommt der Tomcat 4.0 zum Einsatz, welcher mit dem JBoss ausgeliefert wird.[39]

Der Tomcat Webcontainer oder gemeinhin Webserver, verwirklicht die offizielle Referenzimplementierung der APIs der Java Servlets und JavaServer Pages von Sun Microsystems. Das Tomcat Projekt ist Teil eines Großprojekts von Apache mit dem Codenamen Jakarta. Für die Implementierung des Servlet 2.3 API und des JSP 1.2 API in der Version 4 wurde unter dem Namen *Catalina* eine völlig neue Tomcat-Architektur entworfen. Catalina zeichnet sich durch leichte Erweiterbarkeit und hohe

38 siehe http://www.jboss.org, 24.01.05
39 vgl. http://www.wifo.uni-mannheim.de/~mazloumi/seminar/schnaufer-2002-Postgres-SAPDB-JBoss.pdf
40 vgl. hhttp://prdownloads.sourceforge.net/jboss/JBoss.3.0QuickStart.Draft4.pdf, 31.01.05

Modularität aus. Der Tomcat ist von der Apache Software Foundation lizenziert und für die Öffentlichkeit kostenlos.[41]

10.4 IDE Eclipse

Eclipse ist ein Framework, das meist als freie Entwicklungsumgebung genutzt wird. Früher (Version <= 2.1) war Eclipse als erweiterbare IDE gedacht. Seit Version 3.0 ist Eclipse selbst nur der Kern, der einzelne Plugins lädt, die dann die eigentliche Funktionalität zur Verfügung stellen. Sowohl Eclipse als auch die Plugins sind vollständig in Java implementiert und damit nicht nur sprach- sondern auch plattformunabhängig.[42] Eclipse kann ohne zusätzliches Plugin sofort als Java-IDE genutzt werden und Tools wie Ant oder JavaDoc sind sofort verfügbar.

10.4.1 Lomboz-Plugin

Lomboz ist ein kostenloses Eclipse-Plugin für J2EE-Entwickler. Es verwaltet J2EE-Application Server wie JBoss, ermöglicht das Deployment von EJBs und bietet Wizards für das schnelle Erstellen von Servlets, JSPs und alle Arten von EJBs, sowie kleiner Test-Clients. Entwickler können das Start- und Deployment-Verhalten durch einfache Änderungen an entsprechenden XML-Dateien flexibel anpassen. Weiter bietet Lomboz mit seinen Eclipse-Views die Möglichkeit, Container grafisch in Projekten zu überwachen und zu verwalten. Es bietet JSP- und HTML-Code Unterstützung mit Syntax-Prüfung und Syntax-Hervorhebung und die Verfolgung von Fehlern in JSPs mit Eclipse Problem-Markern. Weiter besitzt es Start- und Stopp-Funktionen für nahezu alle J2EE-konforme Application Server und die Verwaltung mehrerer Web- EJB- und Application (EAR)-Module im selben Projekt. Lomboz ermöglicht die Erstellung von Anwendungs-Archiven mit anschließendem Application Server Deployment über sog. Lomboz-Aktionen.[43]

Jedwelche zur Umsetzung der nM-Plattform eingesetzte Software ist für den

41 vgl. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001
42 vgl. http://www.lexikon-definition.de/Eclipse-(IDE).html
43 vgl. http://www.eteration.com/de/Werkzeuge/Lomboz_Details/lomboz_details.html, 26.01.05

privaten, wie professionellen Einsatz frei, wodurch der Firma *netads* keine Kosten in der Anschaffung von Softwarelizenzen entstanden sind, was ein Nebenziel der Diplomarbeit war. Dieser Grundsatz setzte sich auch bei der Erstellung des Diplom-Dokuments fort, welches durchweg mit OpenOffice[44] und JUDE[45], einem freien UML-Tool, entstanden ist.

44 siehe http://www.openoffice.org
45 siehe http://jude.esm.jp

11 Stand und Ausblick

Zum Abschluss der Arbeiten an der nM-Plattform wurde folgender Stand erreicht: Es existiert eine Java-Enterprise-Applikation für mobile und webbasierte Dienste, welche den SMSmonkey als Referenz-Service beinhaltet.
Die entstandene nM-Plattform bietet einen prototypischen Web-Applikations-Teil, der es erlaubt, neue Kunden für den SMSmonkey-Service anzulegen. Anhand des SMSmonkey-MIDlets kann der angelegte Kunde freigeschaltet werden und Standard-SMS versendet und MSGs versendet und empfangen werden. Die Konfiguration ausgesuchter Variablen der nM-Plattform-Applikation, sowie des SMSmonkey-Services ist zur Laufzeit per JMX möglich. Es wurde der Grundstein eines erweiterbaren Sicherheitssystems gelegt, welches den zukünftigen sicheren Betrieb der nM-Plattform und ihrer Services gewährleisten soll.
Anhand der Architektur und Strukturierung, sowie durch die implementierten Schnittstellen der nM-Plattform bietet sich nun die Möglichkeit, weitere mobile und webbasierte Dienste in kürzerer Zeit und mit weniger Aufwand umzusetzen, als wenn die Services in eigenständigen (PHP-)Applikationen realisiert werden würden.
Es wurde ein Dach geschaffen, unter dem eine Anzahl von Services überschaubar, komfortabel wartbar und erweiterbar angesiedelt werden kann, was sich auf Qualität und Sicherheit eines jeden einzelnen Services auswirkt.

Im Zuge einer Weiterentwicklung der nM-Plattform und des SMSmonkey-Services gibt es so manches an Arbeit, was sich im Zeitrahmen der Diplomarbeit nicht unterbringen ließ. Um den SMSmonkey produktiv gehen zu lassen, ist weitere Portierungsarbeit der Funktionalitäten des in PHP gegossenen SMSmonkeys nach dem der nM-Plattform nötig. Zum Beispiel muss der Versand der SMS-Typen *Deluxe* und *Anonym* per MIDlet noch umgesetzt werden. In diesem Zuge ist es auch erforderlich, sich über die restlichen per JMX konfigurierbaren Werte des SMSmonkeys und der nM-Plattform klar zu werden und diese in die Konfiguration einzubauen. Weiter sind Details des Verwaltungs-Systems wie Rabattierung und Kundenwerbung nachzuimplementieren. Allgemein bedarf es der Erstellung eines

Web-Applikations-Konzept zu erstellen. Es muss festgelegt werden, wie der Web-Auftritt, das „Service-Portal“, der nM-Plattform mit mehreren Services auszusehen hat und dem entsprechende Web-Entwicklungsarbeit geleistet werden.
Anhand der durch diese Arbeit errichteten Basis sollte der weitere Ausbau und die Überführung des SMSmonkeys auf die nM-Plattform zügig zu erledigen sein und der Weg dann offen stehen für neue, innovative, mobile Services, welche die Landschaft des *mobile Computings* bereichern werden.

12 Verzeichnisse

12.1 Abbildungsverzeichnis

12.2 Quellen- und Literatur-Verzeichnis

1. Adam Bien, Enterprise Java Frameworks, Addison-Wesley, 2001
2. Andreas Andresen, Komponentenbasierte Softwareentwicklung, Hanser, 2004
3. Mark Wutka, J2EE Delevoper's Guide, Markt&Technik, 2001
4. Turau,Saleck,Schmidt, Java Server Pages und J2EE, dpunkt, 2001
5. Dicken,Hipper,Müßig-Trapp, Datenbanken unter Linux, Mitp, 2000
6. Adam Bien, Entwurfsmuster für die J2EE, Addison-Wesley, 2002

Webframework Struts

7. Javamagazin 04.02, Software&Support, 2002

Neuerungen in J2EE 1.3

8. Javamagazin 03.02, Software&Support, 2002

Servlet Filter

9. http://www.cs.fh-aargau.ch/~gruntz/ courses/sem/ss03/filters.pdf, 20.01.05

Java Management Extention

10. Javamagazin 06.02, Software&Support, 2002
11. http://www.oio.de/public/java/jmx/jmx.html, 30.01.05

J2EE-Entwicklungswerkzeug Lomboz

12. http://www.eteration.com/de/Werkzeuge/Lomboz_Details/lomboz_details.html, 26.01.05

JBoss Dokumentation

13. http://docs.jboss.org/admin-devel/AdminDevelTOC.html, 30.01.05

Datenbanken und JBoss

14. http://www.wifo.uni-mannheim.de/~mazloumi/seminar/schnaufer-2002-Postgres-SAPDB-JBoss.pdf, 30.01.02

JBoss 3.0 QuickStart

15. http://prdownloads.sourceforge.net/jboss/JBoss.3.0QuickStart.Draft4.pdf, 31.01.05

SQL-Injection-Angriffe

16. http://www.infsec.ethz.ch/people/psevinc/sqliadReport.pdf, 31.01.05
17. http://www.net-security.org/dl/articles/IntegrigyIntrotoSQLInjectionAttacks.pdf, 31.01.05

Prepared Statements

18. http://dev.mysql.com/tech-resources/articles/4.1/prepared-statements.html, 31.01.05
19. http://www.galileocomputing.de/openbook/javainsel4/javainsel_20_011.htm, 31.01.05
20. http://dev.mysql.com/tech-resources/articles/4.1/prepared-statements.html, 31.01.05

Java MySQL-Connector

21. http://dev.mysql.com/doc/connector/j/en/index.html, 31.01.05

12.3 Abkürzungen

DBMS	Datenbankmanagement
EJB	Java Enterprise Bean
GPRS	General Packet Radio Service
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JMS	Java Message Service
J2EE	Java 2 Platform, Enterprise Edition
LGPL	Lesser General Public Licence
MDB	Message Driven Bean
MSG	Textnachrichtenformat von netads
MVC	Model View Controller
RMI	Remote Method Invocation
SMS	Short Message Service
SNMP	Simple Network Management Protocol
SQL	Structured Query Language.
UML	Unified Modeling Language
VM	Virtual Machine
VO	Value Object

13 Dank

An dieser Stelle möchte ich mich bei der Firma netads, speziell bei deren Geschäftsführer Tobias Frech bedanken, welcher mir die Möglichkeit gab diese Diplomarbeit im Hause netads zu schreiben. Ansgar Gerlicher sei herzlichst für Vermittlung und Kontaktanbahnung zur Firma netads gedankt. Weiter möchte ich mich bei Herrn Kriha für die Betreuung der Arbeit und für die in deren Zuge geführten inspirierenden und wegweisenden Gespräche bedanken. Den Mitarbeitern der HdM Herrn van der Kamp und Herrn Kuhn ist besonders für ihre Hilfe mit dem Test-Rechner in der FH zu danken. Dank gilt auch meinen Eltern für ihre Unterstützung. Auch den Testlesern der Diplomarbeit sei hier für ihren Mühe und Einsatz gedankt. Besonderen Dank verdient meine Freundin Manuela, welche mir, in der nicht gerade als unstressig zu bezeichnenden Zeit der Diplomarbeit, immer beigestanden ist und diesbezüglich so mache Macke von mir aushalten musste.